DERIVING CLASSIFIERS WITH SINGLE AND MULTI-LABEL RULES USING NEW ASSOCIATIVE CLASSIFICATION METHODS

by

Neda Abdelhamid

Submitted for the degree of Doctor of Philosophy

School of Informatics and Computer Science

Demontfort University

November 2013

DECLARATION

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature _____

ACKNOWLEDGMENTS

First and foremost, I thank God for giving me the strength and ability to pursue this goal in my life.

I thank my supervisors for their support and patience throughout.

A special thanks to Rami and Suhail, for providing me with the feature collection tool that helped me gather the data for my Case Study.

DEDICATION

I would like to dedicate this thesis with my utmost gratitude to my husband, for his continuous cheer on, patience and empathy, he is my true supporter.

My hyperactive teenage kids, Fayez my motivator and Hatem my hero. Love you guys!!

Last but certainly not least, my loving parents for their enduring encouragement and their true faith in me.

There aren’t enough words that could possibly describe what each and one of you all mean to me.

PUBLICATIONS

Most of the chapters in this thesis have been published or submitted for publication in refereed international journals or conferences. The published papers are given below. There are two journal papers under review.

Journals

- Abdelhamid N., Ayesh A., Thabtah F. (2014b) Phishing Detection based Associative Classification Data Mining. Expert Systems With Applications (ESWA). Elsevier.
- Abdelhamid N., Ayesh A. (2014a) Associative Classification Approaches: Review and Comparison (2014c) *Journal of Information and Knowledge Management* (JIKM).
- Abdelhamid N., Ayesh A., Hadi W. (2014c) MCAC: Multi-label Rules Generation via Parallel Associative Classification. Parallel Processing Letters journal Volume 24(2), pp. 1-25.WorldScinet.
- Abdelhamid N., Ayesh A., Thabtah F., Ahmadi S., Hadi W. (2012b) MAC: A multiclass associative classification algorithm. *Journal of Information and Knowledge Management*(JIKM). 11 (2), pp. 1250011-1 - 1250011-10. WorldScinet.

Conferences

- Abdelhamid N., Ayesh A. Thabtah F., Hadi W. (2014d) Emerging Trends in Associative Classification Mining. Proceedings of the ICIIE conference. To be presented in March 2014, Malaysia.
- Abdelhamid N., Ayesh A., Thabtah F. (2013) Classification. Proceedings of the International conference on AI '2013, pp. 687-695. LV, USA. Associative Classification Mining for Website Phishing
- Abdelhamid N., Ayesh A.,Thabtah F. (2012a) An Experimental Study of Three Different Rule Ranking Formulas in Associative Classification Mining. Proceedings of the *IEEE 7th International Conference for Internet Technology and Secured Transactions*, pp. (795-800), London, UK.
- Abdelhamid N., Ayesh A., Thabtah F., Hadi W., Ahmadi S., (2011) A new multiclass associative classification data mining algorithm. Proceedings of the 2011 *ACM* Conference on Innovations in Computing and Engineering Machinery, pp. (24-31). Amman, Jordan. (CICEM 2011)

ABSTRACT

Associative Classification (AC) in data mining is a rule based approach that uses association rule techniques to construct accurate classification systems (classifiers). The majority of existing AC algorithms extract one class per rule and ignore other class labels even when they have large data representation. Thus, extending current AC algorithms to find and extract multi-label rules is promising research direction since new hidden knowledge is revealed for decision makers. Furthermore, the exponential growth of rules in AC has been investigated in this thesis aiming to minimise the number of candidate rules, and therefore reducing the classifier size so end-user can easily exploit and maintain it. Moreover, an investigation to both rule ranking and test data classification steps have been conducted in order to improve the performance of AC algorithms in regards to predictive accuracy.

Overall, this thesis investigates different problems related to AC not limited to the ones listed above, and the results are new AC algorithms that devise single and multi-label rules from different applications data sets, together with comprehensive experimental results. To be exact, the first algorithm proposed named Multi-class Associative Classifier (MAC): This algorithm derives classifiers where each rule is connected with a single class from a training data set. MAC enhanced the rule discovery, rule ranking, rule filtering and classification of test data in AC. The second algorithm proposed is called Multi-label Classifier based Associative Classification (MCAC) that adds on MAC a novel rule discovery method which discovers multi-label rules from single label data without learning from parts of the training data set. These rules denote vital information ignored by most current AC algorithms which benefit both the end-user and the classifier's predictive accuracy. Lastly, the vital problem related to web threats called "website phishing detection" was deeply investigated where a technical solution based on AC has been introduced in Chapter 6. Particularly, we were able to detect new type of knowledge and enhance the detection rate with respect to error rate using our proposed algorithms and against a large collected phishing data set.

Thorough experimental tests utilising large numbers of University of California Irvine (UCI) data sets and a variety of real application data collections related to website classification and trainer timetabling problems reveal that MAC and MCAC generates better quality classifiers if compared with other AC and rule based algorithms with respect to various evaluation measures, i.e. error rate, Label-Weight, Any-Label, number of rules, etc. This is mainly due to the different improvements related to rule discovery, rule filtering, rule sorting, classification step, and more importantly the new type of knowledge associated with the proposed algorithms. Most chapters in this thesis have been disseminated or under review in journals and refereed conference proceedings.

Contents

List of Figures

List of Algorithms

List of Tables

List of Acronyms

AC	Associative Classification
ACCF	Associative Classification based on Closed Frequent Itemsets
ACN	Associative Classifier with Negative
AC-S	Associative Classification over data streams
AC-CS	An Immune-Inspired Associative Classification Algorithm
ADA	An approach for adaptive associative classification
ADT	Association based Decision Tree
AI	Artificial Intelligence
AIS	Artificial Immune System
AIS-AC	Artificial immune system-associative classification
ARC-BC	Association Rule based Categorizer for all Categories
CAEP	Classification by Aggregating Emerging Patterns
CAR	Class Association Rule
CARGBA	Classification based on Association Rules Generated in a Bidirectional Approach
CBA	Classification based on Association Rule
CCS	Complement Class Support
CMAR	Classification based on Multiple Class-Association Rules
CPAR	Classification based on Predictive Association Rules
CLAC	Correlated lazy associative classifier
C-tree	Compact tree

EP	Emerging Patten
FMP	Full Match Pruning
FOIL	First Order Inductive Learner
FP	Frequent Pattern
HCP	High Classify Pruning Method
HP	High Precedence
IREP	Incremental Reduced Error Pruning
L^3	Live-and-Let-Live
LCA	Looking Class Approach
MAC	Multi-Class Based Associative Classification
MCAC	Multi-label Classifier based Associative Classification
MCAR	Multi-class Classification based on Association Rule
MCRAC	Mining Correlated Rules for Associative Classification
MeSH	Medical Subject Headings
ML	Machine Learning
MMAC	Multi-class, Multi-label Associative Classification
MR-ARM	MapReduce Association Rule Mining
NIS	Natural Immune System
NN	Neural Networks
TID	Transaction Identification Number
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
SARC	Statistical Associative Rule Classification
SVM	Support Vector Machine
TC	Text Categorization
UAC	Uncertain Associative Classification
uCBA	Uncertain Classification based Association
UCI	University of California Irvine
WEKA	Waikato Environment for Knowledge Analysis

Chapter One

1.1 Introduction

In the last few years, the numbers of offline and online data sets stored in different business domains have been significantly growing (Song, 2009). Those data sets conceal vital information that can be used by decision makers in their business processes. The way of discovering and extracting the hidden and valuable information from the online and offline data manually by domain experts is extremely hard, time consuming and requires care and experience. This is simply because the available data is normally huge in size and with great dimensionality. Therefore, intelligent software (data mining tools) are utilised to automatically find the useful information from data which grants businesses the confidence in making key decisions. These decisions work for developing and sustaining businesses competitive advantages, which is defined as setting the company a part from its competitors in an industry (Coulter, 2012).

Data mining is a multidisciplinary field consisting of many contributing scientific domains related to computing mainly Artificial Intelligence (AI), databases, and mathematics (statistics and probability) (Witten and Frank, 2002). There are many definitions for data mining, for example (Song, 2009) defined data mining as the process of producing new patterns from large data sets utilising intelligent methods. We have defined data mining as a science that is concerned about revealing unseen information in a user preferred format from data for specific use.

In general, this thesis's main focus is the development of new predictive data mining algorithms based associative classification (AC) to produce useful and accurate knowledge for the decision makers. This chapter highlights the main problems investigated in this thesis, the work motivations and more importantly main contributions.

1.2 Motivations

AC is a rule based classification approach in data mining that applies association rule techniques in finding class association rules (CARs) (Liu et al., 1998; Baralis, et al., 2004). A CAR (Section 2.3.1 - Definition 2.9) is simply an "If-Then" rule that is easily understood by a human and has a conjunction of attribute values in the "If" part (antecedent) and a target class value in the "Then" part (consequent). This classification approach has gained attention in the last decade from scholars in the data mining due to its ability in discovering data insights other classification approaches are unable to detect. This new DATA INSIGHTS (knowledge) help in improving the predictive accuracy of the models (classifiers) according to several experimental studies, i.e. (Yin and Han, 2003; Chien and Chen, 2010; Wang et al., 2011; Wu et al., 2012; Chen, et al., 2012; Costa, et al., 2013; Jabbar et al., 2013).

The learning methodology employed by most AC algorithms tests every single correlation between the attribute value(s) and the class value in the training data set. Though, this may result in redundant rules and if no appropriate pruning is invoked this can cause an exponential growth of rules (Thabtah, 2007; Veloso, et al., 2011). This problem usually happens when the minimum support (*minsupp*) (Defined in Section 2.3.1) is set to a very small value or the input data set is highly correlated. Thus, one of the primary motivations of this thesis is to minimise the number of candidate rules and remove redundant rules without harming the classifier's accuracy rate.

Another important advantage of AC is the simplicity of the output it generates which contains easy interpretable rules (Li et al., 2008). This surely enables decision makers to easily understand and maintain the classifier. Consider for instance, a medical diagnosis system, where symptoms such as coughing, high temperature, blocked sinus, etc, may relate to different types of illnesses "cold", "flu", etc, and are stored in a data set. When a new patient is going to be diagnosed by a physician, the physician utilises the medical diagnoses model to derive the correlations among the patient attributes (age, gender, medical history, etc), the patient current symptoms and the types of illness (class attribute). It would be advantageous if the correlations in the medical diagnoses system are simple rules that easily understood in order to quickly come up with the right diagnoses. The model can also enable the physician to select the right set of rules matching the patient's symptoms and using these with his own experience one can come up with the appropriate diagnoses. Overall, the physician is not interested in a

probability or a complex decision tree since he does not have time nor he is interested in breaking up the complexity of the output.

Furthermore, finding the complete set of classes per rule in AC is a hard task (Thabtah and Cowling 2007; Veloso, et al., 2011; Bouker, et al., 2012). This is mainly due to the fact that the majority of the current AC algorithms discover and extract only the largest frequency class connected with the attribute value in the training data set and ignore all other class labels. Nevertheless, many applications not limited to medical diagnoses described above, online shopping cart, and website classification may require the generation of rules with multiple labels giving both the classifier and decision makers more alternatives to select from. Consequently, the generation of one class per rule could be insufficient especially if there are several classes with an equal or near equal training data representation. It is desirable to find and extract the complete set of possible classes per rule so end-user can make use of them in their related business activities.

Overall, making classification decisions when ignoring that an attribute value may relate to more than one class can lead to the following consequences:

1) A considerable amount of knowledge is kept uncovered in the training data since one class per rule is usually produced. What about the second, third or fourth class labels? What about if there are two or more class labels linked with the rule's body with similar frequencies in the training data set?

2) There will be only one option (class) for both the algorithm and the decision maker when a rule is used during the classification step. Whereas, having all possible classes gives the algorithm multiple options (classes) and provides the decision maker with additional alternatives.

3) In case of a single class per rule, the classification decision of test data will be either a correct classification or a misclassification. Whereas when more than one class is offered, a class weight / probability is assigned to the test data based on the class frequency with the rule's body in the training data set. This seems more fair and legitimate and can improve the predictive performance of the classifier.

Therefore, another motivation of this thesis is to find all class labels per rule from single label data sets to produce classifiers with new type of rules.

Overall, this thesis deals with several issues related to AC in data mining which are discussed in Section 1.4 and briefly summarised hereunder:

1) Improving the current AC steps by
    a. Investigating rule sorting step in order to come up with the right ranking formula.
    b. Proposing a new rule filtering method to minimise the classifier size
    c. Enhancing classifier accuracy by investigating the class assignment step.
    d. Improving frequent ruleitems (Definition 2.7 – Section 2.3.1) discovery step

All previous improvements resulted in a new AC algorithm which has been applied on large numbers of data sets obtained from UCI data repository (Merz and Murphy, 1996) and real applications.

2) Extending AC algorithms to generate classifiers containing multi-label rules by
    a. Developing new learning algorithm for rules with multiple labels.
    b. Measuring the class weight or probability in the multi-label rules.
    c. Applying the new multi-label rules AC algorithm on real application data collection to measure its performance

This thesis is concerned only with single label data sets related to classification in data mining and not the traditional problem of multi-label classification (Read, et al., 2011) which assumes that each training example is linked with more than one class. In this thesis, each training example is linked with just a single class and the term "multi-label rules" refers to rules having more than one class in their consequent that are devised from single label data sets. Section 2.3.1 (Chapter 2) highlights the differences between the two terms.

1.3 Associative Classification Mining

In market basket analysis (Agrawal and Srikant, 1994) such as large supermarkets, e.g. Morrisons or Asda, there are a massive number of customer transactions executed at the different geographical locations. These transactions contain beneficial information that can be explored by the stores managers in making decisions related to product shelving, seasonal sales, and marketing promotions. Generating the useful concealed information from the transactional database can be done in data mining using association rule.

Association rule is considered a descriptive task (Vaithiyanathan, et al., 2012; Ganesh-Kumar 2013) where relationships among products sold in the database are discovered as rules so store managers can successfully make use of them.

In recent years, association rule algorithms have been modified to treat data related to classification problems (Rameshkumar, et al., 2013; Al-Maqaleh 2012) shifting the aim from descriptive to be predictive. This shifting necessitates re-modelling the entire algorithm's life cycle because of the requirement of prediction as well as rule filtering. In classification data such as medical diagnoses, the goal is not descriptive rather predictive to guess the "type of illness" so rules having only the class attribute values in their consequent are the only ones relevant. In other words, the physician is not interested in rules having the patient's features or the symptoms in their consequent which are normally produced if a typical association rule algorithm is applied on this application's data. These new requirements lead to the appearance of AC that combines association rule and classification together to produce classifiers. The role of association rule in AC becomes limited to learning rules that have class value on their consequents (CARs) and discarding all other rules. Once CARs are derived, new phases including rule sorting, filtering and test data classification are imposed.

AC is discussed in depth in Chapter 2 along with common rule based classification approaches. Hereunder, we briefly list the main steps performed by an AC algorithm.

1) Pre-processing (Optional): Discretisation of continuous attributes in the training data set
2) Rule learning: This step consists of two sub-steps
   - Frequent ruleitem discovery: These are attribute values plus class that have frequency above a predefined user threshold named *minsupp*.
   - Rule formation: These are frequent ruleitems that have confidence values above a predefined user threshold called minimum confidence (*minconf*) (defined in Section 2.3.1)
3) Classifier construction: Choosing the most accurate rules after applying all candidate rules extracted in step 2 on the training data. This step also applies rules sorting.
4) Class assignment: Predicting the class of test data using the classifier which has been built in Step (3). In this step, the performance of the classifier is also recorded.

The AC problem along with main definitions related to it, solution scheme, and advantages are given in Chapter 2 (Section 2.3.1).

1.4 Thesis Raised Issues and Research Questions

In this section, different issues related to AC are highlighted. This includes the generation of multi-label rules, the rule pruning phase to minimise the number of rules extracted, and the use of more than one rule in test data class assignment process, are samples of tackled issues. Moreover, the applicability of the proposed algorithms on two critical domains (website phishing classification, the trainer timetabling for a financial institution) is investigated. Next sub-sections discuss these issues.

1.4.1 Multi-label Rules Discovery

One of the challenges in AC is that most current algorithms are unable to generate all class labels associated with an attribute value in the data set. Commonly, an AC algorithm devises only the highest frequency class linked with the attribute value. Nevertheless, there could be more than one class linked with the rule's body in different rows in the training data set with high representation making choosing only one class questionable. For instance, consider attribute value $< x_1, x_2 >$ in a training data of 100 examples and two classes (c_2, c_3). Assume that $< x_1, x_2 >$ are connected with classes c_2 and c_3 ten and nine times respectively. A typical AC algorithm will devise a rule such as $x_1 \wedge x_2 \rightarrow c_2$ and not consider the rule $x_1 \wedge x_2 \rightarrow c_3$ since attribute value $< x_1, x_2 >$ appeared ten times with class c_2 which is only one extra training example than class c_3. Though, class c_3 should be included in the rule rather discarded since it brings up crucial information for the decision maker and has large frequency. So favouring class c_2 over c_3 due to one additional training example is not justified. As a matter of fact, not generating the possible class labels for each rule can be seen as ignoring useful knowledge that can be important to the classifier's accuracy. The research question(s) raised to treat the abovementioned problem is: Would deriving additional useful knowledge (rules) from single label data improve the predictive performance of the classifier?

1.4.2 Improving Classifiers Performance

Different important issues related to enhancing the various steps performed in current AC algorithms have been investigated in this thesis as follows:

- Cutting down the number of rules in the classifier without harming the classification accuracy
- Enhancing the class assignment process
- Improving rule sorting
- Minimising the number of TIDs intersections of ruleitems in the rule discovery step

Hereunder we briefly discuss each issue.

A. Reducing the Classifier Size

One of the main problems associated with AC is that the classifier size derived by this type of algorithms is normally large. The main reason for extracting a huge number of rules is due to the mechanism of inducing the rules which is inherited from association rule where every relationship between an attribute value and the class value is discovered. Thus, we investigated the rule pruning problem particularly after producing the candidate rules and before constructing the classifier aiming to minimise the classifier's final number of rules. The research question(s) we would like to answer for this problem is: Can the number of rules in the classifier be further reduced when evaluating the candidate rules generated after the learning step and how?

B. Test Data Class Assignment and Frequent Ruleitem Discovery

When a test data requires a class, most current AC algorithms seek for the first rule in the classifier identical to the test data and allocate its class to the test data. Nevertheless, there could be more than one rule contained within the test data in the classifier which makes selecting just a single rule inappropriate and unfair decision. In addition, using all relevant rules matching the test data to make the classification will be more legitimate decision simply because a) No single rule preference occurs and b) larger in size rules set is utilised.

Furthermore, mining data sets to find the set of frequent ruleitems is another problem in AC especially when the input data is greatly correlated. Specifically, and during rule learning, the numbers of TIDs intersections performed by disjoint ruleitems is massive and therefore we focus on the rule learning step aiming to answer the following research questions:

Can we cut down the numbers of ruleitems TIDs intersections and how?

C. Rule Sorting Evaluation

Choosing the appropriate rule sorting formula in AC is a critical task that may impact on the selection of the rules during the classification of test data and thus the accuracy of the classifier may get affected. When rules are having high rank in the classifier they are checked first for predicting the test data. Therefore, we want to ensure that rules with high rank have high positive influence on the accuracy. In addition, there could be multiple rules having similar tie breaking criteria (confidence, support, etc) (Definition related to these terms are given in Chapter 2- Section 2.6) during sorting which sometimes forces the algorithm to perform random selection. The research question that we intend to answer for rule sorting is:

Can we reduce the tie breaking among rules during rule sorting to minimise random selection and consequently end up with high quality rules?

1.4.3 Detecting Phishing Websites

Phishing is an online security problem defined as the art of imitating a truthful website for a company in order to steal critical financial information related to online users such as bank account numbers, credit card numbers, passwords, etc, (Mohammad, et al., 2012). Often, the phishy website has content similar to the truthful website in order to deceive online users. Recent studies, i.e. (Gartner, 2011), showed that phishing costs banks and credit card companies in USA billions of dollars per year. Thus, it is crucial to find technical solutions based on learning rules from websites features to minimise its effect. In data mining, phishing is a typical classification problem which involves classifying websites based on their features to one or more of predefined categories. In AC, limited research works have tackled the problem of website phishing detection.

1.4.4 Testing the Proposed Algorithms on Real Data

Different classification data sets related to University of California Irvine (UCI) repository and others to real trainer scheduling application for a financial institution have been collected. The scheduling application is based on "selecting the right local search methods while constructing the trainer timetable" using a general function called the hyperheuristic (Cowling and Chakhlevitch, 2007). For each decision point, the hyperheuristic usually chooses one local search methods (human ways to build a

scheduling solution) that can improve the current solution during the process of constructing the timetable. We intend to investigate the classifiers derived from the real data (schedules) built by the hyperhueristic for the trainer scheduling problem to seek the upsides and downsides of one of our developed AC algorithms (Section 1.5.1).

1.5 Thesis Contributions

In this section, we underline the thesis contributions to each different issue raised in Section 1.4.

1.5.1 Producing Multi-Label Rules

For devising multi-label rules from single label data sets, we propose an algorithm called "Multi-label Classifier based on Associative Classification" (MCAC) that discovers all classes per rule at an early stage without learning from parts of the training data set. Our algorithm sorts the classes within each generated rule based on their weights (probabilities) computed early from the training data set. These new multi-label rules correspond to hidden knowledge that users can make use of by having multiple alternatives (classes) rather than a single one. Further, the classifier can assign the class probabilities associated with the new rules rather than the classes themselves to test data. Chapter 4 elaborates further on MCAC's features. Our multi-label algorithm, (MCAC) has been published in (Abdelhamid, 2014b).

1.5.2 Enhancing the Performance of AC Steps

For the different issues related to enhancing the performance of the current AC steps (Section 1.4.2), we propose a new AC algorithm called "Multiclass Associative Classification" (MAC) which was published in (Abdelhamid, et al., 2011) and an extended version in (Abdelhamid, et al., 2012b). The following contributions are made within MAC:

A. Reducing the Classifier Size

For the classifier construction step, a new rule pruning method has been developed. This method is designed to ensure more training examples coverage per rule without drastically affecting the classifier predictive accuracy. Our rule pruning method has

been published in (Abdelhamid, et al., 2012a) and showed that it reduced the classifier size during evaluating candidate rules against the training data set. Lastly, our rule pruning method reduces overfitting on the training data set by relaxing the class similarity option when the rule is tested on the training data set to seek its potential data coverage. Details on the pruning method of MAC and its results are given in Chapters 3 and 5 – (Sections 3.2.3 and 5.5.2) respectively.

B. Test Data Class Assignment and Frequent Ruleitem Discovery

For test data classification step, a new procedure has been developed based on a set of rules. Our procedure does not require complex mathematical formula to classify test data rather the class represented by the largest number of rules is assigned to the test data. This gives the prediction decision legitimacy because the class belonging to largest number of rules is utilised. Experimental evaluation conducted in Chapter 5 demonstrated that the developed procedure has a positive effect on the accuracy of the classifiers derived from large numbers of data sets. Our procedure has been part of the proposed MAC algorithm and it has been explained fully in Chapter 3 – Section 3.2.4. Lastly, we enhance MAC's classification procedure in MCAC algorithm by merging "single rule" and "group of rules" prediction approaches. The aim is to evaluate at the first instance the best ranked rule matching the test data (Single rule) and when this fails MCAC takes on all rules contained within the test data (Group of rules).

Moreover, for improving the learning step of finding frequent ruleitems (attribute values plus class). We enhanced the TIDs intersections process of ruleitems by imposing a new training method that only intersects disjoint ruleitems TIDs at a given iteration if both of them share the same class. As a consequence, the numbers of items TIDs intersections have been substantially reduced. Experiments in Section 5.5.2 elaborates on this issue.

C. Rule Sorting Evaluation

To improve rule sorting step, we conduct extensive experimentations to evaluate the current parameters relevant to the rule sorting process such as rule's confidence, support, and length. We have selected different data sets from the UCI data repository and implemented different combinations of the rule sorting parameters. The bases of our comparison are the classification accuracy when a certain rule sorting formula is used and the classifier size. The results obtained enabled us to create a new rule sorting formula that has improved the sorting process by reducing rule random selection.

Details on these experimental tests are given in Chapter 5 - Section 5.5.3 and have been published in (Abdelhamid, et al., 2012a).

1.5.3 Detecting Phishing Websites Case Study

We have studied the vital problem of website phishing classification in Chapter 6 extensively by focusing on the set of website's features that may play critical role in determining the type of websites. We have gathered large numbers of phishy and truthful websites from different sources and assessed websites features using two different feature selection methods. Then, our proposed algorithms in this thesis are used to learn important rules that could help in identifying the type of websites. Finally, we evaluated the results obtained on the data set collected and contrasted them with other known AC and rule based classifications according to different evaluation measures. Chapter 6 gives details on the phishing problem, relevant intelligent solutions, data collection, features assessment, and experimentations. The experimental study on the "website phishing classification" has been published in (Abdelhamid, et al., 2013a) and an extended full version in (Abdelhamid, et al., 2014b).

1.5.4 Real Data Experimentations

Over 20 different UCI data sets and multiple scheduling solutions (data sets (over 3200 instances)) have been utilised for evaluating the proposed algorithms and other popular AC and rule based classification techniques. We have applied both MAC and MCAC on the UCI data sets and the MCAC algorithm on the scheduling data sets then compared the results with those obtained from known algorithms in regards to different evaluation measures in Chapter 5. The results revealed that MAC outperforms known AC and rule based classification algorithms with respect to different measures such as accuracy and number of rules. Lastly, MCAC proved to generate multi-label rules missed by current AC algorithms from the trainer scheduling data and derived more accurate classifiers than the other algorithms considered with reference to Label-Weight, Any-Label, and other performance measures.

1.5.5 Review on Associative Classification

AC has attracted several researchers for two main reasons: the simplicity of the outcome (classifier) and its predictive power in forecasting the class of unseen data. Yet, the AC reviews are rare and therefore we conduct a detailed survey on AC so that other scholars can use our analysis as starting point to their research. In this review (Chapter 2), we have focused on the available different methods in the AC literature that are related to data preparation, learning frequent ruleitems, extracting rules, sorting rules, and predicting the class of test data. Particularly, we critically compared the different procedures used by the algorithms at each step of the AC lifecycle and discussed their pros and cons. Part of this review was published in (Abdelhamid, et al., 2014a).

1.6 Thesis Structure

This thesis comprises of seven different chapters where Chapter 2 surveys AC data mining methods. Precisely, it introduces the AC problem; data formats used by the algorithms to represent the training data, rule inducing methods, rule sorting methods, classifier building methods and finally class assignment procedures. We also include in Chapter 2 common learning approaches for rules based classification. In Chapter 3, the first proposed AC algorithm in this thesis is described. It is named MAC for constructing single label rules classifiers. This chapter presents an enhanced ruleitems discovery method, a new rule filtering method that reduces the size of the classifier and an investigation of the rule sorting step. Lastly presented in Chapter 3 is a new class assignment procedure.

Chapter 4 is devoted for the second proposed algorithm in this thesis named MCAC. This algorithm devises multi-label rules from data sets associated with a single class. Chapter 5 introduces the implementation detail of the proposed algorithms, the evaluation measures used and the experiments setting. More importantly presented in Chapter 5 are the experimental tests and their analysis. This includes utilising large numbers of data sets and a wide range of rule based classification techniques to assess the performance of the proposed algorithms. Lastly Chapter 5 evaluates MCAC on real data generated from the scheduling application domain for a financial trainer. Chapter 6 investigates a crucial web application called “Website phishing classification” and then the applicability of MAC and MCAC is tested on data sets generated from this

application. Finally, Chapter 7 summarises the main contributions of the thesis, states the conclusions and recommends new possible research directions related to AC in data mining.

Chapter Two

Associative Classification Mining

2.1 Introduction

Two data mining tasks specifically classification and association rule are correlated in which association rule finds relationships among attribute values in a database whereas classification's goal is allocating class labels to test data. When these tasks get merged the result is Associative Classification (AC) which employs association rule to only discover the rules and adds on top of that additional steps, i.e. (sorting, pruning, and prediction).

Normally, an AC algorithm operates in three main phases. During the first phase, it looks for hidden correlations among the attribute values and the class in the input data and generates them as "Class Association Rule" (CARs) in "IF-THEN" format (Chen and Huang, 2005; Chien and Chen, 2010; Shekhawat and Dhande, 2011; Wu, et al., 2012). After the complete set of CARs are found, ranking and pruning procedures (phase 2) start operating where the ranking procedure sorts rules according to certain thresholds such as confidence and support (Li, et al., 2008). Further, during pruning, useless rules are discarded from the complete set of CARs. The output of phase 2 is the set of CARs which represents the classifier. Lastly, the classifier derived gets tested on new independent data set to measure its effectiveness in forecasting the class of unseen test cases. The output of the classification phase is the classifier's performance in the context of accuracy.

Research studies for instance (Lan et al., 2006; Thabtah, et al., 2010; Pal and Jain, 2010; Hooshsadat and Zaïane, 2012; Wu, et al., 2012) have shown that AC mining has two distinguishing features over other traditional classification approaches. The first one is that it produces very simple knowledge (rules) that can be easily interpreted and manually updated by the end-user. Secondly, this approach often finds additional useful hidden information and therefore the error rate of the resulting classifier is minimised.

There are a number of AC algorithms that have been proposed in the last decade including Classification based Association (CBA) (Liu et al., 1998), Classification based on Predictive Association Rules (CPAR) (Yin and Han, 2003), Classification based on Multiple Association Rules (CMAR) (Li et al., 2001), CAAR (Xu, et al., 2004), Negative-Rules (Antonie and Zaïane, 2004), Live and Let Live (L^3) (Baralis and Torino, 2002), Multiclass Classification based Association Rules (MCAR) (Thabtah et al., 2005), Class based Associative Classification Algorithm (CACA) (Tang and Liao, 2007), Fitcar (Cerf, et al., 2008), Associative Classification Based on Closed Frequent (ACCF) (Li et al., 2008), An Associative Classification with Negative Rules (ACN) (Kundu et al., 2008), Classification based on Boosting Association Rules (CBAR) (Yoon and Lee, 2008), uncertain CBA (uCBA) (Qin et al., 2010) LCA (Thabtah, et al., 2010), ADA (Wang, et al., 2011), Multiclass Associative Classification (Abdelhamid, et al., 2012b), X-class (Costa, et al., 2013) and others. These algorithms employ different methodologies for knowledge reasoning, rule sorting, rule pruning, and class assignment for test data.

In this chapter, we firstly highlight rule based classification in general then the problem of AC is investigated and the different strategies employed in each step by the various AC algorithms are contrasted. Also, advantages and disadvantages of AC and its main differences with other rule based classification approaches are discussed. Lastly, theoretical analysis on rule learning, pruning, ranking and class assignment procedures are conducted.

The rest of the chapter is organised as follows: Popular rule based classification approaches are discussed in Section 2.2. AC problem, its solution scheme, and its main advantages and disadvantages are discussed in Section 2.3. Section 2.4 is devoted to the different data representation models in AC, and the different learning strategies employed in AC are surveyed in Section 2.5. Rule sorting and its associated procedures are surveyed in Section 2.6, and Section 2.7 highlights the different methods employed to build the classifier and to prune unnecessary rules. Section 2.8 sheds the light on prediction methods in AC, and the new emerging trends. Finally conclusions and future research are demonstrated in Section 2.9.

2.2 Rule based Classification in Data Mining

In general, there are a number of tasks in data mining, including, association rule, clustering, classification (Witten and Frank, 2002). To deal with each task, scholars

have developed different algorithms. Classification in data mining is concerned about building a model called the classifier from labelled historical data to guess a target value normally called the class in unseen data. The main goal of classification is to predict the class and that's why it is known as a predictive model. Common applications are credit card scoring, loan approval and medical diagnoses.

A number of different classification approaches have been developed to build classifiers from data such as decision trees (Quinlan, 1993), rule induction (Cohen, 1995), covering (Cendrowska, 1987), AC (Li, et al., 2001), probability (Duda and Hart, 1973), and others. Since this thesis is about AC which is part of the family of rule based classification, we briefly review common rule based classification approaches such as decision trees, rule induction and covering in this section. Another reason for making this section is that some of the algorithms described hereunder are used in the experimental chapter (Chapter 5) for comparison purposes with the proposed AC algorithms (Chapters 3 and 4).

2.2.1 Decision Trees

Ross Quinlan proposed a classification technique called ID3 (Quinlan, 1979) that is able to classify data using a tree. This method can be considered part of rule based classification since the classifier (tree) can be converted into a set of rules where each path from the root node to the leaf denotes a rule. Figure 2.1 shows a tree consisting of 5 rules. The way to build a tree by the ID3 algorithm is based on a mathematical formula called the information gain (IG) (Equation 2.1) where all the attributes IGs in the training data set are evaluated to pick up the root node. The attribute with the highest gain is selected as a root node and a branch for each of its value is constructed. The IG basically evaluates how good is the attribute in splitting the data based on the class labels. The more pure the result of a split using an attribute in terms of class labels the highest gain is given to that attribute. The algorithm repeats the same process for the remaining attributes until the tree cannot be split any more or all data instances in a node are having the same class. Once the tree is finished, a rule will be represented by a path starting from the root to any leaf in which the rule's body is the nodes on the path and the class is the leaf.

After the introduction of ID3 algorithm, Quinlan enhanced it by adding pruning methods in order to simplify the outcome by removing sub-trees that have large error rates. In a brief, the pruning involves computing the error of the sub-tree and compares

it with their leaves. The result of the enhancement on ID3 is an algorithm called C4.5 (Quinlan, 1993) which showed superiority over other classification techniques like probabilistic and covering.

$$\text{Gain}(D, A) = \text{Entropy}(D) - \sum ((|D_a| / |D|) * \text{Entropy}(D_a)) \quad (2.1)$$

where

$$\text{Entropy}(D) = \sum -P_c \log_2 P_c \quad (2.2)$$

where P_v = probability that D belongs to class c.

D_a = subset of D for which A has value a

$|D_a|$ = number of examples in D_a, and $|D|$ = Size of D.

.

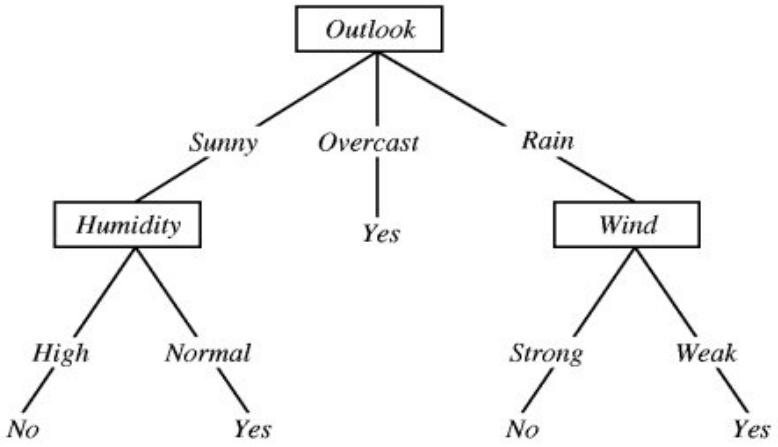

Fig. 2.1 Decision tree for weather data set (Witten and Frank 2002)

2.2.2 Covering Classification

In covering classification (Cendrowska, 1987) like PRISM algorithm, rules are derived in greedy way in which PRISM splits the training data set into subsets with respects to class values. Then for each subset, the algorithm forms an empty rule and searches for the attribute value that has the highest expected accuracy (defined in Equation 2.3) and appends it into the rule body, and continues finding attribute values until the current candidate rule becomes with maximum expected accuracy (often 100%). Once this happens, the algorithm generates the rule and removes all of its positive instances (data in the subset that belong to the rule). The same process is repeated to produce the rest of the rules from the remaining uncovered data in the subset until the subset becomes empty or no rule with acceptable expected accuracy can be derived. At that

point the algorithm moves on to the next class subset and repeats the same process until all rules in all class data subsets are generated and merged to form the classifier.

One notable problem about this classification approach is that rules are found from subsets of the training data and not from the whole data set. This makes local classifiers rather globally ones. Further, the effort required to find the best attribute value to append into a rule at any stage of the learning phase is exhaustive when we have high dimensional training data sets.

$$(P/T) \tag{2.3}$$

2.2.3 Rule Induction

Rule induction approach in classification works similar to covering approach but with more intensive pruning and optimization of the rules set. Often, a rule induction algorithm like RIPPER (Cohen, 1995) divides the training data set with respect to class labels. Then starting with the least frequent class set, it builds a rule by adding items (attribute values) to its body until the rule is perfect (the number of negative examples covered by the rule is zero). For each candidate empty rule, the algorithm looks for the best attribute value in the data set using IG (defined in Equation 2.1) and appends it to the rule's body. It keeps adding attribute values until the rule becomes perfect at that point the rule gets generated. This phase is called rule growing. At the same time while rules are built, RIPPER uses extensive pruning using both the positive and negative examples associated with the candidate rules to reduce rules redundancy and eliminates unnecessary attribute values. The algorithm stops building the rules when any rule found has 50% error or in a new implementation of RIPPER when the minimum description length (MDL) of the rules set after adding a candidate rule is larger than the one obtained before adding the candidate rule.

Another pruning occurs on the candidate rules set to devise the final classifier. So for each candidate rule generated, two substitute rules are made: its replacement and its revision. The first one is made by growing an empty rule r_i' and filtering it to minimise the error on of the overall rules set. The revision the rule is built in similar fashion except the algorithm just inserts an additional item to the rule's body, and examines the original and the revised rule against the data to choose the rule with the least error rate. These extensive pruning in RIPPER explains the small size classifiers generated by this type of algorithms. Experimentations on a number of UCI data sets (Merz and Murphy,

1996) showed that rule induction algorithms such as RIPPER scale well in accuracy rate when compared to decision trees (Cohen, 1995).

2.2.4 PART Approach

A hybrid classification algorithm that uses decision trees and rule induction approaches together to produce classifiers in one phase rather than two phases called PART was proposed in (Frank and Witten, 1998). PART employs rule induction to generate the candidate rules set and then filters this set out using pruning methods adopted from decision trees. PART builds a rule as rule induction algorithms but rather constructing the rule directly from the data, it derives a sub-tree (partial decision tree) from the data and then PART converts the path leading to the leaf with the largest coverage into a rule and the sub-tree gets discarded along with its positive instances from the data set. The same process is repeated until all instances in the data set is removed.

2.2.5 OneRule

One Rule is a simple rule based algorithm that was proposed by (Holte, 1993). This algorithm makes a one-level tree and produces rules that are connected with the most frequent class in the training data set (having the largest data coverage). For all attribute values in the training data set, OneRule iterates over the training data examples and computes the frequency of each attribute value with respect to available class labels. The algorithm selects the most frequent attribute and class and generates them as rule if they pass an error rate check. Finally, the algorithm repeats the same step to generate the subsequent rules until it finds a rule with unacceptable error at that stage the rule discovery process terminates.

2.2.6 Classification based Association / Associative Classification

Classification based association is another name for AC in data mining which is merging of association rule and classification. This approach had come to surface as a promising research discipline in a paper titled “Integrating classification and association rule” (Liu, et al., 1998). In AC, the training phase is about searching for hidden knowledge among the attribute values and the class and then the classifier is constructed after sorting the knowledge and pruning redundant knowledge. Many research studies including (Yin and Han, 2003; Veloso, et al., 2007; Baralis et al., 2008; Li et al., 2008; Ye et al., 2008; Niu et al., 2009; Thabtah et al., 2010; Veloso, et al., 2011; Yu, et al.,

2011; Elsayed et al., 2012; Zhu et al., 2012; Costa et al., 2013; Jabbar, 2013) revealed that AC usually extracts good classifiers with reference to error rate.

The remaining sections of this chapter focus on the different steps performed by AC algorithms in which we comprehensively review:

1) The AC problem, the general solution scheme, and important terms.
2) Data formats.
3) Rule discovery methods.
4) Rule sorting methods.
5) Rule pruning and classifier building methods.
6) Test data classification procedures.

2.3 Associative Classification Framework

2.3.1 The problem Statement and Related Definitions

Given a training data set *D*, which has *n* distinct attributes A_1, A_2, … , A_n and *C* is a list of classes. The number of cases in *D* is denoted |*D*|. An attribute may be categorical (where each attribute takes a value from a known set of possible values) or continuous. For categorical attributes, all possible values are mapped to a set of positive integers. In the case of continuous attributes, any discretisation method can be applied. The goal is to construct a classifier from *D*, e.g. $Cl : A \rightarrow C$, which can forecast the class of test cases where *A* is the set of attribute values and C is the set of classes.

The majority of AC algorithms mainly depend on a threshold called *minsupp* which represents the frequency of the attribute value and its associated class (AttributeValue, class) in the training data set from the size of that data set. Any attribute value plus its related class that passes *minsupp* is known as a frequent ruleitem, and when the frequent ruleitem belongs to a single attribute, it is said to be a frequent 1- ruleitem. Another important threshold in AC is the *minconf*, which can be defined as the frequency of the attribute value and its related class in the training data set from the frequency of the attributes value in the training data. Hereunder are the main definitions related to AC:

Definition 2.1: An *AttributeValue* can be described as an attribute name A_i and its value a_i, denoted (A_i, a_i).

Definition 2.2: The j_{th} *row* or a *training case* in *D* can be described as a list of attribute values $(A_{j1}, a_{j1}), \ldots, (A_{jv}, a_{jv})$, plus a class denoted by c_j.

Definition 2.3: An *AttributeValueSet* set can be described as a set of disjoint attribute values contained in a training case, denoted $< (A_{i1}, a_{i1}), \ldots, (A_{iv}, a_{iv})>$.

Definition 2.4: A *ruleitem r* is of the form <*antecedent, c*>, where *antecedent* is an *AttributeValueSet* and $c \varepsilon C$ is a class.

Definition 2.5: The actual occurrence (*actoccr*) of a *ruleitem r in D* is the number of examples in *D* that match *r's antecedent*.

Definition 2.6: The support count (*suppcount*) of *ruleitem* is the number of examples in *D* that matches *r's antecedent,* and belongs to a class *c*.

Definition 2.7: A *ruleitem r* passes the *minsupp* if, $suppcount(r)/|D| \geq minsupp$. Such a *ruleitem* is said to be a *frequent ruleitem*.

Definition 2.8: A *ruleitem r* passes *minconf* threshold if $suppcount(r) / actoccr(r) \geq minconf$.

Definition 2.9: A single label rule (Class Association Rule) is represented as: $Antecedent \rightarrow c$, where antecedent is an *AttributeValueSet* and the consequent is a class.

Definition 2.10: A multi-label rule *r* is represented as $x_1 \wedge x_2 \wedge \ldots \wedge x_n \rightarrow c_1 \vee c_2 \vee \ldots \vee c_n$ where x_1 is an attribute value and $c_n \subseteq C$, is a class.

Definition 2.11: A single label classifier contains rules in which each one is connected to one class value. Rules like definition 2.9.

Definition 2.12: A multi-label classifier contains rules in which some of them are connected to a set of classes (definition 4.1).

Definition 2.13: An input data is called binary if it has one class attribute with two possible values and each of its training examples is connected to just one class value.

Definition 2.14: An input data is called multi-class if it has one class attribute with more than two values and each of its training examples is connected to just one class value.

Definition 2.15: An input data is called multi-label if each of its training examples is connected to multiple class values (set of classes per training example).

This thesis deals only with single label classification therefore the traditional multi-label classification problem, i.e. (Tsoumakas, et al., 2010; Yang, et al., 2009; Taiwiah and Sheng, 2013) is out of the thesis scope since it is a totally different issue. The difference is that firstly the input data to our model is associated with one class whereas in multi-label classification each training example may associate with many classes. Secondly our classifier contains multi-label rules where each of these is connected with a disjunctive set of classes as given in Definition 2.12 above, whereas in multi-label classification models normally there is no disjunctive rules classifiers. Finally, our classification process depends on rule based classifiers where a single rule is fired to classify a test data. This is unlike most of the multi-label classification algorithms in the literature where a class membership function is employed. For instance, to classify a test data each class in the training data is evaluated and the class or set of classes that are relevant (binary relevance) are assigned to it. More details can be found in (Read et al., 2011).

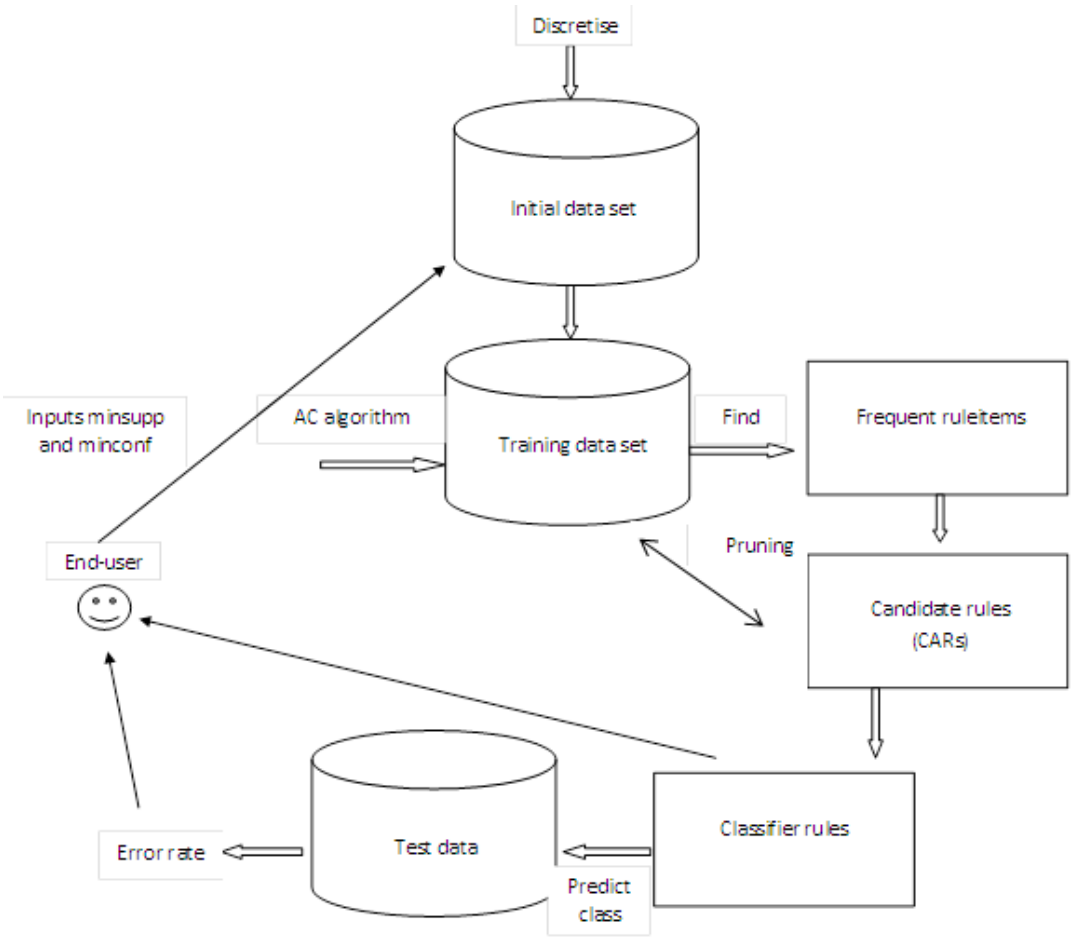

Fig. 2.2 The general life cycle in AC

Table 2.1 Training data set

Row number	Att1	Att2	Class
1	a_1	b_1	c_2
2	a_1	b_1	c_2
3	a_2	b_1	c_1
4	a_1	b_2	c_1
5	a_3	b_1	c_1
6	a_1	b_1	c_2
7	a_2	b_2	c_1
8	a_1	b_2	c_1
9	a_1	b_2	c_1
10	a_1	b_2	c_2

2.3.2 General Solution Scheme in Associative Classification

The majority of AC algorithms operate in three steps, step one involves rules discovery and production, and in step two, a classifier is built from the discovered rules found in step one, and lastly the classifier is evaluated on test data in step three. This is shown in Figure 2.2. To show the process of discovering rules and making the classifier, consider the input data displayed in Table 2.1, which represents three attributes (Att_1, Att_2) and the class attribute (Class). The *minsupp* and *minconf* are assumed to be set given 30% and 50%, respectively. A typical AC algorithm such as CBA firstly discovers all frequent ruleitems which hold enough supports (Table 2.2a). Once all frequent ruleitems are found, then CBA transforms the subset of which hold enough confidence values into rules. The bold rows within Table 2.2a are the rules, and from those the classifier is derived. A rule is put into the classifier if it classifies a number of training examples. Meaning, one subset of the discovered rules is chosen to make the classifier which in turn is evaluated against an independent data set to obtain its effectiveness.

Normally, AC algorithms discover frequent ruleitems by iterating over the training data set many times. In scan (1), frequent 1- ruleitems set is found, and in each later scan, they start with frequent ruleitems discovered in the previous scan (n) in order to derive possible frequent (n +1)-ruleitem, and so on. Once all frequent ruleitems are derived, the algorithms generate the set of candidate CARs from the frequent ruleitems that pass the *minconf* threshold. Overall, the step of generating the frequent ruleitems is a hard task that requires excessive processing because of the possible ruleitems support counting in each iteration (Liu, et al., 2001; Zhu, et al., 2012).

2.3.3 Advantages of Associative Classification

AC has been studied in last decade and applied in different real world application domains including bioinformatics (Clare and King, 2001), medical document categorization (Rak et al., 2005), security (Ye et al., 2008), text categorisation (Abumansour, et al., 2010), and others. The high applicability of this classification approach is mainly due to several advantages offered such as the simplicity of the outputted classifier, the high predictive accuracy of the classifier and the end-user maintenance of the classifier where rules can be easily sorted, added and removed. In this section, we shed the light on the main advantages, disadvantages of AC mining.

Some scholars consider AC a unique case of the association rule mining since it

Table 2.2a Frequent items derived by CBA from Table 2.1

Frequent attribute value	Support	Confidence
<a1>, c2	**40%**	**57.10%**
<a1>, c1	30%	42.85%
<b1>, c2	**30%**	**60%**
<b2>, c1	**40%**	**80%**
<a1,b1>,c2	**30%**	**100%**
<a1,b2>, c1	**30%**	**75%**

produces only the correlations among attribute values and the class attribute in a data set, whereas association rule mining discovers all correlations among attribute values treating the class attribute as any other attribute. For instance, (Liu et al., 1998; Liu et al., 2001; Ye at al., 2008) applied the Apriori algorithm on classification benchmarks and kept only the rules that their consequent contain the class value, and simply ignored the remaining rules. They applied association rule mining in the rule discovery step of AC to derive all possible rules and then they filtered out rules representing correlations among the attribute values. Other scholars consider AC a standalone research branch in classification that at early research stages employed association rule mining in its rule discovery step and then added upon that the classifier construction and class assignment (prediction) steps. Latterly AC mining evolved to use new methodologies for rule discovery other than association rule such as Emerging Patterns (EPs) (Yu et al., 2011), IG (Su et al., 2008), etc. Nevertheless, both sides agreed that AC had its own characteristics.

One of the primary advantages of AC is its ability to discover additional hidden knowledge that other classification approaches are unable to find. The main reason for finding the additional knowledge is the learning methodology employed which tests

every single correlation between the attribute value(s) in the training data set and the class value. This additional knowledge proved to enhance the accuracy of the classifier according to several experimental studies, i.e. (Yu et al., 2011; Qin et al., 2010). Though, the additional knowledge may contain redundant or conflicting rules in which if no appropriate pruning is invoked can cause larger problem called the exponential growth of rules (Li et al., 2001; Thabtah, 2007). This problem usually happens when the *minsupp* is set to a very small value.

Another important advantage of AC is the simplicity of the output it generates which contains chunks of knowledge represented as simple rules. This surely enables the decision maker to easily understand, interpret and maintain the classifier.

2.3.4 Associative Classification vs. Rule based Classification

We can consider AC an approach that follows rule based classification simply because the classifier consists of a set of "If-Then" rules. However, there are differences between AC and other rule based classification approaches mainly in the ways rules are found and test data is assigned the right class. The rules are derived in covering classification such as PRISM algorithm from subsets of the training data set and not the whole set, and the learning strategy is greedy since the algorithm is searching for the largest expected accuracy rule after testing all attribute values in a certain subset. On the contrary, AC exploits the whole training data set once aiming to build a global classifier. Precisely, it finds the set of CARs from the complete training data set.

Moreover, covering algorithms normally derive local classifiers. The extracted rules are said to be local because once a rule is discovered, all cases linked with them in the input data are removed and the process continues until a termination state is met, e.g. the rule discovered has unacceptable error rate. Moreover, the searching for rules in these algorithms is exhaustive since for instance Incremental Reduced Error Pruning algorithm (IREP) (Quinlan, 1993) chooses the rules using the First Order Inductive Learner gain (FOIL-gain). In other words, the rule with the highest FOIL-gain has higher chance to be picked in classification step. Unlike covering and rule induction approaches that require exhaustive search to build local classifiers, AC searches the whole training data set aiming to build a comprehensive classifier. Normally, to construct a classifier in AC, all CARs are first generated and one subset is chosen to

represent the final classifier. This subset is chosen after ranking the rules and invoking certain pruning procedures.

Furthermore, decision trees such as C5 (Quinlan, 1998) derive the classifier as a tree where each path from the top node to the end node (leaf) represents a rule. In this context, one cannot add or update the tree without having large impact on nodes and leaves within the tree. This necessitates reshaping the complete decision tree to reflect changes that might occur. Alternatively, if the end-user wishes to insert a new rule in a classifier produced by an AC algorithm he can do that in a straightforward manner without affecting the rules set. Table 2.2b depicts the general differences between AC and other rule based classification approaches with reference to rule learning methodologies, classifier output format, and other criteria.

Table 2.2b General differences between AC and other rule-based classification approaches

Classification Approach Name	Rule Discovery Methodology	Ranking Methodology	Pruning Procedure	Output format
AC	Association Rule Mining	confidence, support, rules generated first	database coverage, lazy pruning	Rules
Decision Tree	Etropy & Information Gain	No ranking	Backward and forward pruning, e.g. pessimistic error	Trees
Covering	Greedy	No ranking	Some algorithms use expected rule's accuracy, Foil-gain, and others no pruning	Rules
Rule Induction	Exhaustive Search	Certain mathematical measure, e.g. (Foil-fain)	Reduced error Pruning, Incremental REP	Rules

2.4 Data Representation in Associative Classification

2.4.1 Horizontal and Vertical

Before the dissemination of the MMAC algorithm in 2004 (Thabtah, et al., 2004), there was only one data representation in AC adopted from association rule discovery called horizontal (Liu, et al., 1998). In the horizontal data format, the training data set consists of a number of cases or rows in which each row has a number followed by the list of attribute values. Table 2.1 that has been displayed earlier is an example of horizontal data format. The authors of MMAC have introduced the vertical data format in AC where the training data set gets converted into a table similar to Table 2.3 in which each attribute value is represented by its locations (row numbers) in the training data set. This representation is highly effective as we will see later particularly in computing the support for each attribute value. Therefore, on the contrary of the horizontal data format which is often associated with computational costs such as the time required for merging disjoint ruleitems, and ruleitems support calculation, the discovery of frequent

ruleitems in the vertical data format is accomplished by simple intersections of disjoint attribute values locations.

For example, the determinations of frequent 2-ruleitem are based on intersecting disjoint frequent 1-ruleitem locations. For instance, the candidate 2-ruleitem $<(Attr_1,a_1),(Attr_2,b_1), c_2>$ in Table 2.3 can be evaluated to determine whether it is frequent or not by intersecting the locations of ruleitems $<(Attr_1,a_1)>$ and $<(Attr_2,b_1)>$, respectively. In other words, the set (1,2,4,6,8,9,10) is intersected with the set (1,2,3,5,6), and the results of the intersection (1,2,6) denotes the row numbers in the training data in which the new candidate ruleitem $<(Attr_1,a_1),(Attr_2,b_1), c_2>$ has appeared. Then by locating the row numbers of the class "C_2" we simply find out that this candidate 2-ruleitem size, i.e. 3, denotes the support count. If the support count is larger than the *minsupp* threshold then this candidate 2-ruleitem will become frequent, otherwise it will be discarded.

In vertical data format, the support values of candidate ruleitems of size *m* can be easily computed by simple intersecting the locations of the disjoint ruleitems of size (*m*-1). Thus, the locations of ruleitems hold vital information related to attribute values in the training data set that are relatively simple and easy to maintain within any kind of data structure, and consequently there is no need for repetitive data scan to calculate the support of new candidate ruleitems at any iteration. This saves training time as well as it reduces I/O overhead (Zaki and Gouda, 2003; Tang and Liao, 2007).

Table 2.3 Vertical data representation of Table 2.1

$(Attr_1, a_1)$	$(Attr_1, a_2)$	$(Attr_1, a_3)$	$(Attr_2, b_1)$	$(Attr_2, b_2)$	$(Class, c_1)$	$(Class, c_2)$
1	3	5	1	4	3	1
2	7		2	7	4	2
4			3	8	5	6
6			5	9	7	10
8			6	10	8	
9					9	
10						

2.4.2 Line and Item Space

Recently, a new distributed association rule algorithm called MapReduce Association Rule Mining (MR-ARM) (Thabtah and Hammoud, 2013) introduced the idea of using both the horizontal and vertical data representation models interchangeably (line and item spaces representation). This new data format can be easily used in AC algorithms to discover frequent ruleitems, extract rules, sort rules and filter out the candidate rules to build the classifier during the algorithm's life cycle. Precisely, MR-ARM maps each

training case to a unique number (RowId) that denotes the rows locations where the case appears. This RowId will be part of the main identification code of the rules items (attribute values) that first occurred at this row within the training data set. So, every frequent item identification code has two parts: Column ids, and RowId

ItemId = (Column ids) RowId

Column Ids: are the attributes numbers in the training data set which make up this item.

RowId: The row number the item first appeared in the training data set.

When the training data set is transformed into ItemId layout, all transitional data used by the mining algorithm keep the same data layout through the life cycle of the algorithm. The consequence is that the rule generation step becomes simple during the execution process of the algorithm. Another upside of the ItemId data format is the size of intermediate data communicated while the algorithm is operating in the distributed environment is minimised. Below is an example to transform the data inside Table 2.4 to line and item space format based on another example in (Hammoud, 2010) which is further explained by us in Section 2.5.10.

Table 2.4 : Initial data set

RowId	Attributes			Class Label
0	X	Y	Z	L_1
1	Z	Y	Z	L_1
2	Z	W	Z	L_2
3	Z	W	Z	L_1
4	X	Y	X	L_2
5	X	W	X	L_1
6	Z	W	X	L_1
7	Z	Y	W	L_1
8	X	Y	X	L_3

Table 2.4.1 line space format (Hammoud, 2010)

Line:Class	Attributes		
0:0	(0)0	(1)0	(2)0
1:0	(0)1	(1)0	(2)0
2:2	(0)1	(1)2	(2)0
3:3	(0)1	(1)2	(2)0
4:2	(0)0	(1)0	(2)4
5:3	(0)0	(1)2	(2)4
6:3	(0)1	(1)2	(2)4
7:3	(0)1	(1)0	(2)7
8:3	(0)0	(1)0	(2)4

Table 2.4.2 item space format (Hammoud, 2010)

Attribute	Line:Class
(0)0	0:0, 4:2, 8:3
(0)1	1:0, 2:2, 3:3, 6:3, 7:3
(1)0	0:0, 1:0, 4:2, 7:3, 8:3
(1)2	2:2, 3:3, 5:3, 6:6
(2)0	0:0, 1:0, 2:2, 3:3
(2)4	4:2, 5:3, 6:3, 8:3
(2)7	7:3

MR-ARM is the first distributed algorithm that applied the “Line and Item” space data layout to mine big data. Table 2.4.1 is an example of line space layout resulted from the transformation of the data in Table 2.4.1. On the other hand, in “item space” data layout

each item is represented in a data structure that maps the class labels with equivalent rows for this item. In other words, row numbers that the item's classes have appeared.

2.5 Learning Approaches in Associative Classification

The first step in AC mining is about discovering and generating the CARs therefore we can decompose it into two sub-steps (1) the discovery of frequent ruleitems, and (2) the rule generation. Many scholars (Veloso, et al., 2007; Cerf et al., 2008, Niu, et al., 2009; Costa, et al., 2013) consider this step the most challenging steps since it requires significant search, computations, and may necessitate multiple training data set scans. For instance, the CBA algorithm scans the input data *n* times where *n* denotes the number of iterations required to find the complete set of frequent ruleitems. Generally, there are different learning methodologies in AC many of which are adopted from association rule discovery such as Apriori level-wise search (Agrawal, R., and Srikant, 1994), frequent pattern growth (Han, et al., 2000), vertical (Zaki and Gouda, 2003), closed frequent itemsets (Zaki and Hsiao, 2002), and others. Further, there are other learning approaches that are adopted from rule induction and decision trees such as IG based (Su, et al., 2008), FOIL-gain (Yin and Han, 2003) and statistical ones (Jabez, 2011). In this section the different rule learning approaches in AC are surveyed in details.

2.5.1 CBA based Approaches

Apriori is an association rule discovery algorithm that has been proposed in (Agrawal and Srikant, 1994) and its name is based on the fact that it uses prior knowledge of frequent itemsets. A frequent itemset is an item that has a frequency in the input database above the user *minsupp* threshold. The complete set of frequent itemsets are utilised to produce the association rules, and more precisely any frequent itemset that holds enough confidence get converted into a rule. In Apriori, frequent itemsets is found in repeated search, where in each iteration, a database scan is essential to produce the new candidates from frequent itemsets devised in the prior iteration. Apriori uses the "downward-closure" property to minimise search space of the candidate itemsets by cutting down their size during each iteration.

One of the first research studies that showed the utilisation of Apriori in solving classification benchmarks is CBA. This algorithm implements the Apriori

generate_candidate function to find and produce the frequent ruleitems. The main difference between an itemset and a ruleitem is that the ruleitem consists of attribute value plus the class value (<attributes, values>, class), whereas the itemset may be looked at as just an attribute value by itself. Once CBA finds the complete set of frequent ruleitems, then a subset of which passes the *minconf* threshold is converted into CARs.

Since CBA employs Apriori in its learning step, it has inherited some of Apriori's deficiencies especially the repetitive data set scans and the exponential growth of rules (Li, et al., 2001). In particular, since Apriori tests all correlations among the items in the transactional database in the learning step in order to find the rules, the expected numbers of candidate itemsets are often massive. This definitely leads to the generation of large numbers of association rules, and in some cases especially with very low *minsupp* the numbers of rules are in the orders of tens or hundreds of thousands, which consequently limit their use in practical applications. So, after the dissemination of CBA, several AC algorithms have been proposed to overcome some of CBA's deficiencies that have been inherited from Apriori. For instance, CBA (2) was disseminated to overcome the problem of not generating CARs for minority class labels in the training data set (The class balancing issue) (Liu, et al., 2001). Further, CMAR algorithm was developed to improve the searching for frequent ruleitems and it introduced a compact data structure to achieve this goal. Lastly, LCA algorithm (Thabtah, et al., 2010) was developed to minimise the number of candidate itemsets which usually consumes time.

Currently, there are several AC algorithms that uses CBA style during the learning step to find frequent ruleitems and generate the CARs including CBA (2) (Liu, et al., 2001), ARC-BC (Antonie and Zaïane, 2002), NegativeRules (Antonie and Zaïane, 2004), lazy associative (Baralis, et al., 2004), CAAR (Xu, et al., 2004), Entropy associative (Su, etr al., 2008), and ACN (Kundu, et al., 2008). These algorithms have improved upon CBA in one or more of its main steps including rule's (learning, sorting, pruning) or prediction. For example, ARC-BC has been applied on unstructured textual data collection, and lazy AC algorithms such as L^3G (Baralis, et al., 2004) have enhanced the accuracy of CBA by producing more knowledge. Lastly, ACN and Negative rules have discussed the issue of deriving not only positive knowledge but also knowledge with negation in the antecedent or consequent part of the rule. More precisely, ACN was proposed to mine a relatively large set of negative association rules

and then uses both positive and negative rules to build a classifier. A positive rule is of the form $X \Rightarrow Y$ where X, Y are a set of items and X∩Y=ϕ. A negative rule is of the form $X \Rightarrow Y$ where in addition to being a set of items, X or Y will contain at least one negated item.

Lastly, an AC algorithm based neural network (NN) (Shekhawat and Dhande, 2011) adopted CBA rule learning method to discover frequent ruleitems and to extract the candidate rules. Then, the candidate rules are used to as an input for the back propagation algorithm to assign the class of the incoming test instance.

2.5.2 Charm based Approach

A closely related approach to Apriori learning style that reduces the number of candidate itemsets and improves the searching for frequent itemsets called closed itemset was proposed in (Li, et al., 2008). An itemset is said to be closed if none of its immediate supersets has similar support value as that of the itemset. For instance, if {ice, juice, crisps} is an itemset with a support value of 5, and all of its supersets have support values less than 5, then {Ice, Juice, crisps} becomes closed itemset. Normally, closed itemsets are able to answer common inquiries like "is a particular itemset frequent?" and, if so, "what its support value in the input database?". One of the common algorithms for mining closed itemsets is Charm (Zaki and Hsiao, 2002). Unlike Apriori algorithm that usually explores the itemset and the transactional database spaces Charm only explores the itermset space after the first iteration. Moreover, it introduced an efficient candidate searching method that skips many levels of the data structure (itemset tree) to quickly discover the frequent closed itemsets, instead of having to enumerate many possible subsets.

Few years ago, (Li, et al., 2008) extended Charm to handle classification benchmarks in an AC algorithm called ACCF. In particular, ACCF employed the concept of closed itemsets of Charm to cut down the number of CARs produced so that decision makers can control the classifier and edit the rules. Experimental results against eighteen different data collection downloaded from the UCI (Merz and Murphy, 1996) indicated that ACCF produced similar classifier with respect to accuracy to CBA.

2.5.3 Combinatorial Mathematics

One recent AC approach for mining CARs which is based on the theory of combinatorial mathematics was proposed in (Pal and Jain, 2010). The basic idea behind this AC algorithm comes from generating all possible combinations of attribute values in the input data set which is represented as a bitmap and then counting the occurrences of each element within the produced combinations. A combination is just an unordered set of a unique size consisting of a number of elements (attribute values). To clarify the concept of generating the possible combinations of elements for set S, let's assume that S = (X,Y,Z). The possible number of combinations for S can be computed as $2^{|S|}$ and in this case $2^3 = 8$, and shown as (Φ ,X,Y,Z,XY,XZ,YZ,XYZ). Now, the authors have enumerated each element using binary representation so element "X" is represented as 100 and element "XYZ" is represented as 111. The algorithm works in two steps where in step (1) it computes the support value for each combination to generate the candidate ruleitems (attribute value, class value) and then in step (2) it builds the classifier by converting any ruleitems having confidence value larger than the *minconf* into CAR.

This simple rule learning strategy which is based on combinatorial mathematics is not novel since association rule mining algorithms such as Apriori is also based on binary representation of the items within the transactional database and uses efficient pruning method based on the downward closure property to reduce the search space for rules. The AC algorithm presented in (Pal and Jain, 2010) has been tested only on one single data set from UCI repository called "TicTac" which questions its applicability. Lastly, the efficiency of such algorithm was not evaluated especially on high correlated classification data sets which we expect the number of combinations of attribute values to be numerous.

2.5.4 Imbalanced Class Distribution

The classes in some classification data sets are unevenly distributed. This may result in the production of very small number of rules and in some cases no rules at all for the low frequency class and numerous numbers of rules for the high frequency class(s) (Arunasalam and Chawla, 2006). This problem normally happens because of the *minsupp* threshold which controls the rule discovery step in which if we set it to a value larger than certain class frequency there will be no rules representation for that class in the classifier, and several strong rules will be simply ignored. Therefore, researchers have investigated the possibility of utilising multiple supports (Liu, et al., 2001; Baralis

et al., 2008) or other measures such as Complement Class Support (CCS) (Arunasalam and Chawla, 2006) that overcome the class imbalance issues in classification benchmarks.

One possible solution to the class imbalance problem is the abundance of the *minsupp* threshold from taking any role in the rule generation and the use of new measures such as CCS that primarily takes into account positively correlated rules as shown in the equation below:

$$\text{CCS for a rule (R) (A} \longrightarrow \text{C)} = Support(A \cup \bar{C}) / Support(\bar{C}) \qquad (2.4)$$

Where A is the conjunction of the attribute values in R's body and $\bar{C}$ represents the complement of class C. The learning approach of (Arunasalam and Chawla, 2006) only looks for strong correlation between the rule antecedent (rule body) and consequent (class), meaning rules that have low CCS are produced and other rules with high CCS are discarded. Experimentations against 8 data sets from UCI repository showed that CCS algorithm performed well on imbalanced data sets when it comes to predictive accuracy.

Another possible solution to the class imbalance problem is the enhancement performed on CBA algorithm in (Liu, et al., 2001) that considers the frequency of class labels in the input data set, and assigns each class different support value. In other words, the original *minsupp* value is distributed to each class according to the class frequency in the input data set. So, a low frequency class gets a low *minsupp* to guarantee the production of rules for it. An evaluation study against a number of data sets from UCI repository showed that on average the error rate of CBA (2) is lower than that of CBA and C4.5 algorithms.

(Baralis et al., 2004) proposed a related multiple supports approach that looks at the current rules generated for all class labels in iteration *N* in order to amend the support value for class labels that have no rules representation by lowering their support requirement. Therefore, ensuring rules appearance for most of classes in the training data set.

2.5.5 Intersection based Approach

To reduce the number of passes over the input database in horizontal mining algorithms, the Eclat algorithm has been presented in (Zaki, et al., 1997), which requires only a single database scan, addressing the question of whether all frequent itemsets can be

derived in a single pass. Eclat introduced the concept of vertical database representation in association rule (Table 2.3) in which frequent itemsets are found by intersecting tid-lists of candidate itemsets. A tid-list of an item is a simple data structure that contains the locations in which the item has appeared in the training data set.

In 2003, a change of the Eclat has resulted in an algorithm called dEclat which was proposed in (Zaki and Gouda, 2003). The dEclat algorithm uses a newer layout called diffset that avoids storing the complete tids of each itemset; rather the difference between the class and its member itemsets are stored. Two itemsets share the same class if they share a common prefix. A class represents items that the prefix can be extended with to obtain new class. For instance, for a class of itemsets with prefix x, $[x] = \{a_1, a_2, a_3, a_4\}$, one can perform the intersection of xa_i with all xa_j with j>i to get the new classes. From $[x]$, we can obtain classes $[xa_1] = \{a_2, a_3, a_4\}$, $[xa_2] = \{a_3, a_4\}$, $[xa_3] = \{a_4\}$.

In AC mining, (Thabtah, et al., 2004; Thabtah, et al., 2005) modified the tid-list intersection learning used in association rule to handle classification benchmarks in an algorithm called MCAR. This algorithm operates in three steps where in step one complete candidate rules set is discovered and extracted from all attribute values having enough support. Any attribute value plus class (ruleitem) that holds a support less than the *minsupp* gets deleted. The algorithm then tests the extracted rules set in step two to identify rules that have high confidence. These rules form the classifier and all remaining candidate rules are removed in this step. The final step involves testing the classifier on test data to derive its performance.

The first step of MCAR necessitates going over the input data to compute the support of the complete set of ruleitems which are in fact the input to derive the candidate rules set. In particular, the algorithm identifies firstly frequent 1-ruleitems, from those it keeps only those that have hold sufficient support. During going over the input data, the transaction locations connected with each frequent 1-ruleitem is saved in a data structure along with the classes and locations in the training data set. Then, MCAR determines the remaining frequent N ruleitems by joining frequent ruleitems of size N-1. Lastly, once the complete frequent ruleitems set is identified their confidence are calculated to generate those that survive the *minconf* to generate the candidate rules. Experimentations on a number of UCI data sets showed that MCAR outperformed CBA and other classic classification algorithms such as RIPPER and C4.5 with respect to accuracy.

In (Tang and Liao, 2007), a vertical AC algorithm called CACA was proposed. It scans the training data set, stores data in vertical data format like MCAR, counts the frequency of every attribute value and arranges attributes in descending order according to their frequencies. Any attribute which fails to satisfy the *minsupp* is removed in this step. The frequent attribute values are split according to the available classes, and for each class, CACA intersects frequent attribute values locations to cut down the searching space. Any ruleitem resulted from an intersection with enough confidence is saved into a tree data structure as a rule. Results produced by (Tang and Liao, 2007) showed that CACA is competitive to MCAR algorithm on a limited number of UCI data.

2.5.6 Causal, Incremental and Emerging Pattern

The majority of AC algorithms employ *minsupp* and *minconf* which are mainly statistical correlation parameters to discover the rules. The *minsupp* is used mainly to capture frequent attribute values (items) and the *minconf* is hired to show the strong rules from the set of frequent attribute values. A different AC approach based on the idea of causality and Emerging Pattern (EP) has been proposed in (Yu et al., 2011) (Dong et al., 1999). Most of the current AC algorithms determine the correlation between rule antecedent (attribute value) and consequent (class) based on support and confidence parameters. Though correlation only reveals statistical association between a set of objects in an implication, e.g. $X \rightarrow Y$. If we discover causal correlation between the rule antecedent and consequent one can reveal consequential factors with reference to classes in the data set. Therefore, unlike current AC algorithms which mainly produce a large search space for frequent ruleitems during the rule discovery, the use EP can minimise the search space of the candidate ruleitems by only keeping ruleitems that have causal impact on the class. In other words, when CARs are discovered the only attribute values considered in the CARs are those that belong to this causal attribute values space instead of the combinations of all attributes values. This may minimise the demand on resources including training time and memory in the rule discovery step.

The first algorithm which employed causal concept and EP was proposed in (Dong and Li, 1999) and it is called Classification by Aggregating Emerging Patterns (CAEP). An EP is an attribute value that its support changes from a data set to another, with change rate larger than a constant υ. The support rate between two data sets for a given attribute value is called the growth-rate, which can be computed as follows:

$$\frac{Support_{d'}(att)}{Support_{\bar{d}}(att)} \quad (2.5)$$

Where *att* is the attribute value, d and $\bar{d}$ are the data sets which the attribute value's support has changed. Given a *minsupp* threshold and a growth-rate, the algorithm finds EPs that survive υ, also known as υ-attribute values. In mining EPs, the input data set is first divided into parts based on the classes, and a production of all υ-attribute values from one part to another is implemented. Experimental studies (Yu et al., 2011; Dong et al., 1999) showed that EP's based AC algorithms generate competitive classifiers with respect to classification rate if compared to CBA and C4.5 algorithms.

(Wang, et al., 2011) proposed an AC called ADA that constructs rules from both the input training data set as well as the classified resources such as the training data set, current classification rules, and test cases. Meaning the classifier gets amended on the fly after the classified resources reach to a certain amount. The authors have used a co-training method (Mei, et al., 2006) to accomplish the task of updating the classifier by refining the new discovered knowledge by the existing classification rules. The co-training method used in ADA has been adopted from the semi-supervised learning of pattern context where the labelled training documents are exercised to figure out the class labels of the unlabelled cases. More details can be found in (Mei, et al., 2006). Overall, ADA can be considered a semi-incremental AC algorithm since few training cases or users set of frequent patterns (keywords) are only necessary to build the classifier instead of the complete training cases. Then, the classified cases as well as the classification rules are employed to update the classifier by adding or removing rules.

2.5.7 CMAR and Lazy based Approaches

In AC mining, a modified version of the FP-Growth has been successfully implemented by a number of algorithms including Malware detection AC (Ye et al., 2008), L^3G (Baralis, et al., 2004; Baralis, et al., 2008), L^3 (Baralis and Torino, 2002) and CMAR (Li, et al., 2001). Particularly, the first AC algorithm that employed FP-Growth is CMAR which saves the rules in a prefix tree data structure known as a C-tree. The C-tree holds the rules in a descending order according to the rule body support value in the training data set (frequency of the attribute values in the antecedent of the rule). Once a rule is extracted, it gets inserted into the C-tree as a path from the root and its support, confidence and associated class are saved at the last node in the path. When a new rule is about to be inserted into the tree and that rule contains common attribute values with

another already existing rule in the tree, the path of the existing rule is extended to reflect the addition of the new rule.

In 2002, an AC algorithm called L^3 has employed CMAR learning strategy in rule generation, though this algorithm adds on CMAR the concept of lazy pruning. The lazy pruning approach is discussed in Section 2.8.4. Recently, (Ye et al., 2008) have evaluated the applicability of AC on the malware security benchmark problem. Malware is a general term that corresponds to all kinds of unwanted software's like Trojans, spyware, viruses, and others. Since the detection of malware id is challenging especially from large data sets, the authors have adopted the CMAR in order to improve the performance involving the searching for correlations between the security attributes and the class. Experimentations using a number of Windows PE files. i.e. (benign executable) and (malicious executable), have been used to evaluate the algorithm. The results revealed that this algorithm usually achieves the highest identification of malware if compared to decision tree (Quinlan, 1993).

Recently, an AC algorithm was proposed in (Veloso, et al., 2011) which allows both training and testing data to play a role in determining the right rules used for assigning the class to the test case. So, attribute values in the training data that are similar to the test data attribute values are the only one used to learn the rules that in turn are used to assign the right class to the test data delaying the rules reasoning or the training step and merging it with the classification step.

In general, most of the AC algorithms that employ CMAR learning strategy reduces the use of the intermediate usage as well as the searching time for frequent ruleitems if contrasted with CBA-like algorithms (Li, et al., 2001). Experimental studies, i.e. (Ye, et al., 2008), on Spyware data demonstrated that CMAR-like algorithms produce efficient classifiers. Nevertheless, one major deficiency of these algorithms is that the C-tree may not fit in the main memory in cases when the input data is dense and huge in size.

2.5.8 Greedy based Approach

A learning strategy called FOIL that produces rules for each class in the training data set was produced in (Quinlan and Cameron-Jones, 1993). FOIL learns the rules greedily based on a measure called FOIL-gain. The algorithm generates the rules as follow: for each available class *L*, it splits the training data into two subsets, one that contains all cases associated with *L* (positive cases) and one that holds all other cases

associated with the rest of the classes (negative cases). Then FOIL initiates an empty rule (e.g. If empty then *L*), and iterates over the available attribute values to compute the FOIL-gain for each attribute value belonging to *L*, it selects the attribute value with the largest FOIL-gain and adds it in the rule's antecedent. The sample process is repeated until the constructed rule length reaches a certain value and the negative case set is not empty. Once the rule is constructed, all associated positive cases that belong to the attribute value and class *L* are removed. FOIL continues building rules for class *L* until all positive cases are covered (removed), once that occurs it considers another class and repeats the same process until all classes are considered.

The key to success in FOIL learning strategy is the Foil-gain measure which is about assessing the information gained for a particular rule after adding an attribute value to its body. The FOIL-gain measure for a certain attribute value (A_1, v_1) can be calculated using the class information in the training data set. So, for class *L*, the positive cases associated with it are denoted $|P'|$ and the negative cases of *L* are denoted $|N'|$. Once (A_1, v_1) is added by FOIL into a rule *R*, there will be $|P|$ positive and $|N|$ negative training cases that correspond to *R*: $(A1, v1) \rightarrow c$.

$$\text{Foil-gain } (A_1, v_1) = |P| \left(\log_2 \frac{|P|}{|P| + |N|} - \log_2 \frac{|P'|}{|P'| + |N'|} \right). \tag{2.6}$$

It is clear that FOIL always looks for the largest FOIL-gain attribute value in order to add it into the rule. Though, there could be more than one attribute value with similar FOIL-gain which makes the selection of just one attribute value questionable. This also can lead to deterioration in the classification accuracy during the prediction step since a limited number of rules are often extracted. Another problem associated with FOIL learning fashion is that the rules are derived from parts of the training data set and not from the complete set, which makes them local rules and not global ones.

In 2003, (Yin and Han, 2003) proposed an AC algorithm called CPAR that enhances FOIL rule learning in which once a rule such as *R* is constructed, CPAR does not discard the positive cases associated with *R* instead weights of these cases are lowered by a multiplying factor. This enhancement guarantees the production of more rules as a training case is allowed to be covered by multiple rules instead of a single, and consequently the classification accuracy gets improved as well. Moreover, CPAR finds all attribute value with the largest FOIL-gain rather than just one as in FOIL so it

can add multiple attribute values into the rules and thus building multiple rules simultaneously.

Furthermore, the searching process for the attribute value with the largest FOIL-gain can be exhaustive and require storage resources (e.g. main memory) especially when the available number of attributes in the training data set is large. In this context, CPAR adopts an efficient data structure called PNArray (Gehrke, at al., 1998) to keep all necessary data about the rule such as the positive and the negative cases before adding the attribute value into the rule antecedent and after adding it into the rule. It has been shown that CPAR is highly competitive with reference to predictive accuracy to other AC algorithms such as CBA and traditional classification algorithms such as IREP and C4.5 against the UCI data collection.

The different steps in AC mining have been studied in (Chen, et al., 2005) in order to come up with a new algorithm that can take an advantage from the previous studies. The outcome was an algorithm called Mining Correlated Rules for Associative Classification (MCRAC) that learns the rules using FOIL-Gain measure, and then it discards detailed rules and weakly correlated rules similar to CMAR algorithm with minor modifications. Evaluation using ten UCI data sets and using known AC algorithms including CBA, and CPAR showed that MCRAC is competitive to these algorithms in terms of accuracy.

2.5.9 Distributed MapReduce

MapReduce (MR) is a parallel programming model that recently attracted scholars in large enterprises because of its effectiveness and efficiency (Zhao, et al., 2009; Wu et al., 2009). Though, not much research on simulating the performance of MapReduce has been done. Recently, MapReduce has been adopted by many search enterprises such as Google, and Amazon to enable building petabyte data centers comprising hundreds of thousands of nodes. These data centers are of low cost hardware and with a software infrastructure to allow parallel processing analysis of the stored data (Dhok and Varma, 2010). MapReduce model provides a software infrastructure to simplify writing applications that can access and process this massive data. However, the cluster setup to get optimum performance is not a trivial problem. It needs configuration of tens of setup parameters and dynamic job parameters which affect every task execution.

In AC mining, the first distributed algorithm called MR was proposed in (Hamoud, 2010). This algorithm has four steps that may utilise one or multiple distributed bases MapReduce jobs:

- Data Initialising: Transforming the training data into (ColumnId) RowId layout.
- Rule Generation: Discovering frequent ruleitems and extracting the rules.
- Building the classifier: Choosing the best rules to make the classifier from all candidate rules found in Step (2).
- Classifying test data: Single rule method is used to allocate the appropriate class label to the test data.

In the first step, the training data set is transformed into ItemId format as described in Section 2.4.2, and all intermediate data produced during the algorithm execution keep the same data layout. The discovery step of frequent ruleitem in each level of the MR algorithm repeats the transformation of the intermediate data between the Line-space and the Item-space until the complete set of frequent ruleitems are devised.

For instance, giving the training data set in Table 2.4 with support count = 2, and the last attribute in that table corresponds to the class label. The MR algorithm transforms the Table 2.4 into Line-space (Table 2.4.1) and maps the data to values in the Item-space format. So each item in the Line-space is called to give out the list of "ItemId, (Line,Label)".

```
(line 0) <0:0, (0)0, (1)0, (2)0> ToFrequentItem.Mapper   <(0)0 ,(0:0)>,  <(1)0, (0:0)>,  <(2)0, (0:0)>.
(line 1) <1:0, (0)1, (1)0 , (2)0> => ToFrequentItem.Mapper =><(0)1,( 1:0)>,< (1)0  ,( 1:0)>,< (2)0 ,( 1:0)>
... etc.
```

For example, for items "x" and "z", their relevant data given to the Reducer method are,

```
<(0)0, 0:0 >,<(0)0, 4:2 >, <(0)0, 5:3 >,<(0)0, 8:3 >  "Reducer" < (0)0 ,[ 0:0, 4:2, 5:3, 8:3]>
.....  "Reducer" <  (0)1 ,[ 1:0, 2:2, 3:3, 6:3, 7:3]>
```

Items (0)0 and (0)1 are frequent ruleitems correspond to "x" and "z" with frequency 2, and 3, respectively with class label "L_3", i.e. third class value. The output here will be

```
(0)0    { sup=2 , conf=0.500,     0:[0]     2:[4]     3:[5, 8]}
(0)1    { sup=3 , conf=0.600,     0:[1]     2:[2]     3:[3, 6, 7]}
(1)0    { sup=2 , conf=0.400, 0:[0, 1]      2:[4]     3:[7, 8]}
(1)2    { sup=3 , conf=0.750,  2:[2]        3:[3, 5, 6]}
(2)0    { sup=2 , conf=0.500,  0:[0, 1] 2:[2]         3:[3]}
(2)4    { sup=3 , conf=0.750,  2:[4]        3:[5, 6, 8]}
```

frequent 1-ruleitems. The frequent 1-ruleitems appearances are transformed into the Lin-space data layout using the MR. So <"x", L_3> and <"y", L_3> frequent ruleitems Line-space are:

```
(0)0    { sup=2 , conf=0.500,        0:[0]    2:[4]    3:[5, 8]} => "Mapper" =>
                    <0:0, (0)0>, <4:2, (0)0>,<5:3, (0)0>,<8:3, (0)0>
(0)1    { sup=3 , conf=0.600,        0:[1]    2:[2]    3:[3, 6, 7]} =>"Mapper" =>
                    <1:0, (0)1>, <2:2, (0)1>,<3:3, (0)1>,<6:3, (0)1>,<7:3, (0)1>
```

The outputs are grouped by the row numbers and given to the "Reducer" that collect the ItemIds and derives them to line-space. Next step, MR discovers frequent N-ruleitems by merging frequent N-1 ruleitems.

2.6 Multi-label Rules in Associative Classification

Few research attempts have tackled the problem of generating multi-label rules from single label data, e.g. MMAC (Thabtah, et al., 2004). MMAC extracts rules with multiple labels in a separate step named the recursive learning. Other than this algorithm existing AC algorithms produce one class per rule in the classifier and thus we can consider them single label classifiers based algorithms. In the searching process for rules in the training data set during learning step, these algorithms only consider the largest frequency class associated with the attribute value and produce it in the potential rule consequent. It should be noted that this section contains AC algorithms that either derive class set per rules or utilise more than one class in classifying test data from classification data sets associated with a single label.

2.6.1 MMAC

The first AC that considers the production of multiple labels in the rule consequent is MMAC (Thabtah, et al., 2004). It is a repetitive learning algorithm that derives N single label rules from parts of the training data set similar to other AC algorithms and then merges all of them to produce the multiple labels rules in a separate phase called the recursive learning. This phase requires multiple scans for parts of the training data set and thus may demand high resources when the input data is large.

To be exact, the recursive learning phase of MMAC algorithm combines local classifiers derived during a number of iterations into a multiple labels classifier. For a given training data set *T*, MMAC extracts the first single label classifier in iteration one.

Then all training cases associated with the derived rules are discarded, and the remaining unclassified cases in the original training data set comprise a new data set T_1. In the next iteration, the algorithm finds all rules from T_1, builds another single label classifier, removes all cases in T_1 which are associated with the generated rules, and so forth. The results are *n* classifiers in which MMAC merges them to form a multi-label rules.

Four notable drawbacks are associated with this algorithm:

1) It can be said that the rules connected with multiple class labels in the classifier has been constructed from parts of the training data set similar to covering algorithms like FOIL and PRISM rather than the complete training data set.
2) The recursive learning phase of the MMAC algorithm is a separate step that requires repetitive scanning of parts of the training data set. To be more precise, MMAC is in fact invoked *N* times executing all required steps (frequent ruleitem discovery, rule extraction, rule sorting, rule filtering, test data prediction) on certain parts of the training data which is a burden especially when the input data is large or the candidate numbers of rules are massive. So for instance, if we end up with rules associated with four class labels this means that MMAC has been executed 4 times.
3) Single label rules derived at iteration (2) and its successors are associated with biased support since their actual support values are calculated from parts of the training data set rather than the complete data set.
4) When a multi-label rule is formed by the MMAC there is no defined method for computing its support. When a new multi-label rule is created there should be a way of giving it a new support value. MMAC basically gives any multi-label rule the support belonging to the last single label rule appended to it. Though, this is not correct because more than one support values are associated with the multi-label rule one for each of its class label. This biased support which is given to the multi-label rule by MMAC may impact the rank of rule in the classifier and therefore the classification accuracy may get directly impacted as a result.

2.6.2 Correlated Lazy Approach

An AC classification algorithm called Correlated Lazy Associative Classifier (CLAC) (Veloso, et al., 2007) that adopts lazy classification and delays the reasoning process

until a test data is given was developed in 2007. Though, CLAC does not allow the presence of multiple classes in the consequent of the rules. The learning strategy used by CLAC assigns a gain value consisting of the confidence and support value of the rule(s) having the class and belonging to the test case, and then one class applicable to the test case is used in the prediction. Specifically, CLAC gives the test case the class with the largest gain, and considers the test case a new feature and iteratively assigns new class labels to the test case until no more labels can be found. Empirical evaluations showed that CLAC achieved good performance in regards to one-error rate on three textual data sets if compared to BoosTexter algorithm (Schapire and Singer, 2000).

An enhancement on CLAC was proposed in (Veloso, et al., 2011) claiming that deriving all candidate rules in the training phase could be problematic in cases when the *minsupp* is set to low values. Thus, suggesting using the test data attribute values as valuable information to reduce the search space of applicable rules. This approach may require searching in the training data set to learn the rules every time a test data is about to be classified which can be problematic. Lastly, in cases when the relevant subset of the training data to the test case are insufficient (they fail the support) this algorithm considers the default class which may increase error rate.

A domain specific algorithm based on association rule named CLAC that uses more than one class to categorise MEDLINE documents was proposed in (Rak, et al., 2005). This algorithm uses keywords from MeSH and the documents content (title, abstract) to process the input textual document and converts it into a training instance consisting of meaningful keywords and a category. Mesh is a controlled vocabulary thesaurus of medical keywords containing over 22000 descriptors in 11 levels hierarchical structure. Once the data set is processed, the algorithm employs items reoccurrences during the rule learning and classification steps. The novelty of this approach is the classification process in which the authors have developed different methods to assign an incoming document all its relevant categories. The main method used to classify test examples is based on the relevant rules confidence values without rule sorting. No multi-label rules in the classifier is involved in this domain specific algorithm.

Experimentations using the OHSUMED medical text collection (Rak, et al., 2005) on the different class assignment procedures (described above) have been conducted in regards to classification accuracy and classifier size. The results indicated that the new

classification method produced acceptable accuracy results though it has generated larger number of rules.

2.6.3 Rank Label

An AC algorithm called Rank-Label (Thabtah and Cowling, 2007) was proposed to deal with the problem of class ranking in the rules extracted. This algorithm proposed a post training method that adjusts the classes rank in each of the rule inside the classifier. The majority of AC mining algorithms use the classification rules learned from the training data set for constructing the classifier which in turn is applied to predict the class of unseen test data. Though, in circumstances where there are limited input data or the input data gets frequently updated, there should be a mechanism that can take into consideration 1) the new update(s) on the source data and the classified resources (rules and the predicted test data).

Moreover, the problem of correlation between the class and the training cases may result in generating rules associated with wrong class since these rules overlap in the training cases. Precisely, the rule discovery strategies employed by current AC algorithm are normally adopted from association rule mining in which these algorithms allow the training case to be covered by multiple rules. So when a rule is inserted into the classifier and its related data are removed from the training data set other candidate lower ranked rules sharing with the inserted rule its training cases may get affected. Thus, the support and confidence for these impacted rules may change because of the training cases removal. This problem makes classes associated with many rules not the most accurate ones and require re-ranking of class labels.

The post training method of Rank-Label adjusts the position of the classes in the rules taking into consideration the rules overlapping in the training cases. This heuristic operates as follow: Starting with the top ranked rule, it iterates over the training data set removing all training cases applicable to the rule. Then, the support and confidence of the lower ranked rules that share with the selected rule its training cases get decreased. This may result in adjusting the classes position(s) in the lower ranked rules and the largest frequency class for some of these rules may not be the fittest class any more. The process is repeated until all training data cases are removed or the algorithm has iterated over all rules. This process is similar to covering approach in classification in which it allow the training case to be covered by just a single rule in the classifier. Empirical study on UCI data sets showed that removing the overlapping among the rules in the classifier slightly enhance the performance in accuracy when compared to

CBA, and C4.5 algorithms. One drawback of this post training method is that the rules are learnt from parts of the training data set which may involve computing biased support and confidence values.

2.7 Rule Sorting Procedures

In classification algorithms there should be a way to favour one rule over another during the classification process of test data, and thus rule sorting plays a critical role. For instance, decision tree algorithms like C4.5 (Quinlan, 1993) have a clear preference in their searching for the best attribute decision node, which is, the attribute selection method based on IG.

In AC mining, an algorithm uses a rule sorting procedure to distinguish rules in which it gives high confidence and support rules higher ranks. This is crucial since usually rules with higher rank are tested first during the predicting of test cases, and the resulting classifier accuracy depends heavily on rules used during the prediction phase. There are several different criteria taken by scholars in AC when sorting rules. For instance, CBA based algorithms consider the rule's confidence and support as main criteria for rule favouring, and CMAR adds on that the rule length when two or more rules have similar confidence and support values. Further, (Su, et al., 2008) have employed IG in rule preference in which a rule is said to be informative if it has a gain above a certain threshold. In this section, we highlight different rule sorting procedures in AC.

2.7.1 Confidence, Support and Cardinality

The first rule sorting procedure in AC was introduced in (Liu, et al., 1998) and it is based on rule's confidence, and support. This procedure is displayed in Figure 2.3. Using this rule preference procedure there will be huge number of rules with similar confidence and support values. Consider for example two data sets ("Auto" and "Glass") from the UCI data repository. Assume that the *minsupp* and *minconf* are set to 2% and 40%, respectively. If we apply a common AC algorithm such as CACA or MCAR, the number of discovered rules with identical confidence from the "Auto" and "Glass" data sets are 2660 and 759 respectively without rule pruning. When we apply the rule support as a tie breaking condition, we end up with 2492 and 624 rules with similar confidence and support values. This example if limited show clearly a direct evidence that in AC mining there are great number of rules that have common

confidence and support and thus more tie breaking conditions are needed to minimise the chance for rule arbitrary choices.

There are a number of AC algorithms that employ the rule sorting procedure shown in Figure 2.3 including CBA (2), CARGBA, ACCF, CAAR, and others. In 2005, MCAR algorithm adds rule's class distribution in the training data set as tie breaking condition beside the rule confidence, support, and antecedent length. In particular, if two rules have identical confidence, support, and antecedent length, MCAR favours the rule which is associated with the class that has larger frequency in the training data set.

Given two rules, R_1 and R_2, R_1 precedes R_2 if

1. The confidence of R_1 is larger than that of R_2
2. The confidences of R_1 and R_2 are the identical, but the support of R_1 is larger than that of R_2
3. The confidence and support of R_1 and R_2 are the identical, but the R_1 generated before R_2

Fig. 2.3 CBA rule sorting method

2.7.2 Specific Rule

Lazy AC algorithms such as L^3 often prefer rules that hold large number of attribute values in their antecedent. These kinds of rules are named specific rules. In fact, lazy algorithms try to hold almost all knowledge discovered even if redundancy exist aiming to maximise the predictive power of the classifiers. Unlike CBA, the L^3 ranking procedure (Figure 2.4) mainly prefers specific rules over general ones in order to give the specific rules higher chance in the prediction step since they are often more accurate than general rules. In the prediction phase, when the specific rules are unable to assign a class to the test case, then general rules, those with smaller number of attributes in their antecedent, are considered.

Given two rules, R_1 and R_2, R_1 precedes R_2 if

1. The confidence of $R_1$is larger than that of R_2
2. The confidences of R_1 and R_2 are the identical, but the support of R_1 is larger than that of R_2
3. The confidence and support values of R_1 and R_2 are the identical, but the R_1 contains more number of attributes in its antecedent than that of R_2

Fig. 2.4 Live-and-Let-Live (L^3) rule sorting method

2.7.3 Information Gain

IG is a mathematical measure mainly used in decision trees to decide which attribute goes into a root and represents the expected amount of information required to determine which class should be given to a new unclassified cases. In other words, it measures how well a given attribute divides the training data cases into classes. The attribute with the highest information is chosen.

Decision tree algorithms such as C4.5 compute IG to assess which attribute goes into a decision node. As described earlier in Section 2.2.1, C4.5 makes the selection of the root based on the most informative attribute and the process of selecting an attribute is repeated recursively at the so-called child nodes of the root, excluding the attributes that have been chosen before, until the remaining training data cases cannot be split any more. At that point, a decision tree is outputted where each node corresponds to an attribute and each arc to a possible value of that attribute. Each path from the root node to any given leaf in the tree corresponds to a rule.

An AC mining which utilises IG as the main criteria for rule sorting was disseminated by (Su, et al., 2008). Specifically, the IG of the rule $r : Cond \longrightarrow C$ is defined as $Gain(r) = G_D - G_{cond} - G_{\overline{cond}}$.

(2.7)

Where

G_D represents the IG of the training data set *D* and is defined as

$$G_D = -\sum_{i=1}^{m} \left|\frac{C_i}{D}\right| \log_2 \left|\frac{C_i}{D}\right| \tag{2.8}$$

Where

$|C_i|$ represents the number of data cases which belong to class C_i.

The IG of the rule antecedent (G_{cond}) is defined as

$$G_{cond} = \frac{N_1}{|D|}\left[-\frac{N_{11}}{N_1} \log_2 \frac{N_{11}}{N_1} - \frac{N_{12}}{N_1} \log_2 \frac{N_{12}}{N_1} \right] \tag{2.9}$$

Where

$$N_1 = |D| * \frac{Support(R)}{Confidence(R)},\ N_{11} = |D| * Support(R), \text{ and } N_{12} = N_1 - N_{11}$$

(2.10)

Finally, the training cases that don't match the rule antecedent are also considered as:

$$G_{\overline{cond}} = \frac{N_2}{|D|}\left[-\sum_{i=1}^{m} \frac{|C_i|}{N_2} \log_2 \frac{|C_i|}{N_2} \right] \text{Where } N_2 = |D| - N_1$$

(2.11)

So the rule (r) is said to be informative if r has support and confidence greater than the *minsupp* and *minconf* as well as the GAIN (r) > 0. After the rules are discovered, the ranking procedure will be invoked where rules with larger gain are placed at a higher rank. In cases when two or more rules have similar gain, then the algorithm evaluates the confidence, and support similar to CBA rule preference procedure.

2.7.4 Discussion on Rule Sorting

Rule sorting is considered a pre-processing phase in AC mining which impact the 1) classifier building process and 2) predicting of test cases. As matter fact, without rule sorting the algorithm will not be able to easily choose the rules that can be employed in the prediction step. Rule preference has been defined differently by AC algorithms. CBA and its successors considered confidence and support the main criteria for rule preference, and MCAR adds upon CBA rule length and the rule's class distribution of the rules if two or more rules have identical confidence and support. On the other hand, unlike CBA and MCAR, L^3 algorithm prefers specific rules over general ones since they contain multiple general rules. Lastly, recent algorithms consider information theory based measures such as IG as the base for rule preferences.

An experimental study (Thabtah et al., 2005) revealed that using confidence, support and rule antecedent cardinality in rule ranking is effective approach. Though, recent studies and the example discussed in Section 2.7 showed that imposing more parameters beside confidence and support may reduce the chance of randomisation in ranking which consequently limits the use of default class later on in prediction step. Finally, approaches that favour specific rules may sometimes gain slight improvement in accuracy, however it suffers from holding large number of rules many of which are never used and thus consuming memory as well as training time.

2.8 Rule Pruning Methods

Once the complete set of candidate rules are found in the training phase and ranked, the AC algorithm has to decide the way it should choose a subset of highly effective rules to represent the classifier. Different means are utilised to make the classifier in AC, for example, CBA and its successors use the database coverage where after testing the candidate rules on the training data set rules that cover correctly a at least one training example is selected and any rule that has no data coverage is deleted. L^3 and L^3G algorithms employ lazy pruning that stores primary and secondary rules in the classifier. Moreover, (Thabtah, et al., 2010) proposed a pruning method based on partial rule matching of the rule body and the training case. This section discusses the different pruning methods including also mathematical ones such as Pessimistic Error Estimation, Chi-Square testing, and others. Yet, firstly we define two important terms related to pruning in AC.

Definition 2.16: A rule is said to fully match a training case if the attribute values in the rule body are contained in the training case.

Definition 2.17: A rule is said to partially match a training case if at least one of the attribute values in the rule body is contained in the training case.

2.8.1 Database Coverage Pruning

The database coverage is the first pruning method in AC that has been applied by CBA to select the classifier and it works similar to rule learning in coverage algorithms like PRISM. This method tests the candidate rules on the training data set in order to select accurate rules for the classifier. Figure 2.5 depicts the database coverage method in which each candidate rule is evaluated on the training data set and when the rule covers training examples it will be saved in the classifier and all of its related training examples are discarded. When a rule is unable to cover a training example the rule is removed.

Input: The complete set of discovered rules *R* sorted, and the training data set *D*
Output: Classifier *CR*

1 Iterate over *R* starting with r_i
2 Selects all examples in *D* relevant to r_i's body
3 If r_i correctly covers an example in *D*
4 Save r_i into the *CR*
5 Delete all examples in *D* covered by` r_i
6 end if
7 If r_i cover no examples in *D*
8 Discard r_i
9 end if
10 end
11 If *D* contains uncovered examples
12 Make a default rule
13 Label the least error rule in R as a cutoff rule.
14 end if

Fig. 2.5 The database coverage method (Liu, et al., 1998)

This method stops when no data is left in the training data set or all candidate rules are tested. In cases when examples are left uncovered they will be employed to make the default rule that denotes the class with largest count in the unclassified training examples. This default rule is applied in the classification step when no rule in the classifier is relevant to the test data. After proposing CBA, several AC algorithms have successfully employed the database coverage in building the classifier, i.e. CBA (2), ARC-BC, CAAR, ACN, and ACCF.

2.8.2 High Classify Pruning

This rule evaluation method (Figure 2.6) (Abumansour, et al., 2010) goes over the complete set of rules after ranking and applies each rule against the training data set. If the rule covers (partially matches) a training case and has a common class to that of the training case it will be inputted into the classifier and all training cases covered by the rule are removed. The method repeats the same process for each remaining rule until the training data set becomes empty, and it considers the rules within the classifier during the prediction step. The distinct difference between this method and the database coverage is that a rule is inputted into the classifier if it partially covers at least one training case, whereas in the database coverage, a rule body must fully match the training case in order to be part of the classifier.

Another similar pruning method called HP (Abumansour, et al., 2010) inserts a rule into the classifier if its body partly contained in a training example without class similarity. The difference between HP and HCP methods is that in the HP, a rule gets added into the classifier if it partially covers at least one training case regardless if it classifies that case correctly or not. On the other hand, in the HCP, a rule must classify a training case correctly in order to be considered in the classifier.

Input: Given a set of generated rules *R*, and training data set *T*
Output: classifier (*Cl*)

```
1  R' = sort(R);
2  For each rule r_i in R' Do
3        Find all applicable training cases in T that partially match r_i's condition
4       If r_i correctly classifies a training case in T
5            Insert the rule at the end of Cl
6            Remove all training cases in T covered by r_i
7       end if
8       If r_i cannot correctly cover any training case in T
9          Remove r_i from R
10      end if
11 end for
```

Fig. 2.6 HCP rule evaluation method

2.8.3 Lazy Pruning

Lazy AC algorithms, limit discarding rules while building the classifier to those that incorrectly cover training examples. The reason behind the idea of limited rule pruning by lazy AC scholars is because they claim that pruning of rules should be restricted to rules having wrong classification on the training data set. Lazy classifiers categorize rules in the classifier into two main groups. The first group contains all candidate rules that have covered correctly training examples, and the second group consists of rules that have no data coverage.

Algorithms that use lazy pruning like L^3 and L^3G consider the rules in the first group (primary) and when no rules are able to cover the test data the second rules group (Secondary) is used. The main distinct difference between the database coverage and lazy pruning is that the second group of rules is not considered by the database coverage. This is an upside for CBA over L^3 particularly for application data that

require a controllable number of rules so the end user easily understand and update the classifier.

Empirical studies, i.e. (Baralis, et al., 2004; Baralis, et al., 2008) using UCI data showed that lazy algorithms may slightly improve the accuracy but usually we end up with huge classifiers which consequently restrict their use.

2.8.4 Long Rules Pruning

A method that prefers rules with a small number of attribute values and, have high confidence was developed in the CMAR (Li, et al., 2001). This method eliminates rules duplication by removing long rules that may contain smaller other rules, and therefore the classifier size get minimised. This pruning method is invoked when a candidate rule is about to be saved into the classifier in which the already inserted rules are checked, if a classifier rule contains the candidate rule it will be discarded and vice versa. After the publication of CMAR algorithm, other AC methods that have adopted this pruning are ACS, Negative classification algorithm and ARC-BC.

2.8.5 Mathematical based Pruning

Pessimistic error was utilised firstly in classification by decision trees algorithms to cut sub-trees and substitute them by leaf nodes because they may cause high error. This is called sub-tree pruning, and the sub-tree error is calculated according to equation 2.12 below.

$$q(v) = \frac{N_v - N_{v,c} + 0.5}{N_v} \tag{2.12}$$

where

N_v is the number of training cases at node *v*

$N_{v,c}$ is the number of training cases belonging to the largest frequency class at node *v*.

The error rate at sub-tree *T*,

$$q(T) = \frac{\sum_{l \in leafs(T)} N_l - N_{l,c} + 0.5}{\sum_{l \in leafs(T)} N_l}. \quad (2.13)$$

The sub-tree T is pruned if $q(v) \leq q(T)$.

CBA was the first AC algorithm that used pessimistic error in which for a candidate rule, the algorithm discards one of its attribute in its body to make a new possible rule. CBA computes the expected error of both rules and substitutes the original rule with the new rule only when the new rule's error is smaller than that of the original rule.

2.8.6 Discussion on Rule Pruning

Pruning in AC is crucial for the successful use of the algorithm on real domains because of the excessive number of rules derived by this family of algorithms. Overall, the reduction of the number of candidate rules and the classifier size has definite advantages that can be briefly summarised as follows:

1) It enables the end-user of controlling the classifier and understanding its rules easily. Think of a general practitioner who has a medical diagnoses system consisting of just fifteen rules compared with another practitioner who has a fifty rule classifier taking into account that the performance of both medical diagnoses systems are not substantially different.
2) It may enhance the classifier construction process, which requires passing over most of the candidate rules and for each training example to figure out rules that have data coverage. The rules that cover training examples are in fact the classifier rules.
3) It may improve the searching process of the right rule to use for classifying a test case during the classification step. When a test data is about to be classified, normally current AC algorithms must iterate over the classifier rules to choose the rule that can classify the test data.

We have seen in the previous sections that there are few different pruning methods proposed in the AC literature mainly depend on the database coverage presented in the CBA algorithm or have been adopted from mathematical measures. For instance, uCBA, LCA, L^3G, L^3 and MCAR slightly amend the database coverage but as CBA necessitate full rule body matching with the training instance besides class similarity.

These excessive pruning procedures limit rule data coverage as a result producing large classifiers. CBA overcomes this by further reducing the candidate rules early using Pessimistic Error pruning method. There is also new attempts toward partial matching of rules with training instances have also been proposed. Nevertheless, this kind of pruning has extremely large training data coverage per rule because a rule is inserted into the classifier during evaluation phase when any of its items is matching any item in the training data. This surely makes the rule cover very large number of data and therefore we end up with limited size classifiers with low accuracy rate.

We believe that a balance between the two pruning approaches to end up with controllable size classifier is a potential research direction in rule pruning. So rather having methods that extremely conservative (tied conditions) like database coverage based ones or too loose like partial matching we proposed a method in Chapter 3 that keeps rule body similarity with the training instance but neglects class similarity condition to reduce overfitting the training data. More details on this method can be found in Section 3.2.3.

2.9 Class Forecasting Methods

This section describes prediction methods in AC that use multiple rules to assign the class of test data.

2.9.1 One Rule Class

CBA algorithm has introduced this classification method (Figure 2.7) in which the rules in the classifier are used starting from the highest ranked rules to allocate the test data the appropriate class. To be exact, the algorithm goes over the classifier rules in descending order and picks the first rule class and allocates it to the test data. This means only the first rules marching the test data classifies it. When no rules is applicable to the test data CBA fires the default class rule and allocates it to the test data. Other AC algorithms that utilise CBA's classification method are L^3, L^3G, LCA, etc.

2.9.2 Predictive Confidence

In the classification step, confidence is the main criteria considered for selecting the rule to assign for test cases. Though, some scholars like (Do et al., 2005) which proposed the AC-S algorithm claims that confidence values for rules are calculated using the training

examples and this is not enough taking into account the existence of test examples. Therefore, there should be other criteria beside confidence to use during the classification step to distinguish among the classifier set of rules. The authors of (Do et al., 2005) introduced the concept of rule predictive confidence which corresponds to the rule's average expected accuracy on the test data case.

Input: Classifier (*CL*), test data (*Ts*), Data structure *Temp*
Output: accuracy *e*

Given a test data set (*Ts*), the classification process works as follow:

```
1 For each test data  Do
2 For each rule r in CL Do
3    Select the first rule that is contained in ts body  and copy it to Temp
4     If Temp contains a rule Do
5              allocate r's class to ts
7           end if
8      else assign the default class to ts
9    end if
10 empty Temp
11 end
12 end
13 compute the accuracy (e) on Ts;
```

Fig. 2.7 CBA prediction method

For example, assume that there is a test data (*ts*) and a classifier rule (r): $Antecedent \rightarrow c$. Further, assume that there is *X* test data examples from *ts* matching *r's* antecedent and having class *ts* and *Y* test data examples matching only *r's* antecedent. Now, when *r* is applied on *ts r* will classify *X* test examples with expected accuracy of (*X*/*Y*) which corresponds to the predictive accuracy of *r*. This measure classifies test data based on information from the rules and the available examples in the test data. Empirical experiments showed that AC-S algorithm is competitive to known AC algorithms like CMAR and CBA.

Input: Classifier (*CL*), test data (*Ts*), array *Temp*
Output: Accuracy *e*

Given a test data (*Ts*), the classification process works as follow:

```
1 For each test data Do
3 For each rule r in CL  Do
4   Find all rules that partly contained in ts and insert them in Temp
5     If Temp contain rules Do
6        split rules in Temp based on class into clusters
7         calculate the average confidence for rules in each cluster
8         allocate the largest cluster confidence class to ts
9     end if
10   else allocate the default rule to ts
11    end if
12    end
13    empty Temp
14      end
15    compute the total number of errors of Ts;
```

Fig. 2.8 Confidence based classification method

2.9.3 Multiple Rules Class

The single rule classification methods described earlier work fine when only one rule matches the test data. Nevertheless, when multiple rules are contained in the test data, decision based on a single rule is inappropriate since the selection of one rule to make the class assignment becomes questionable. Thus, utilising all rules that match the test case for class prediction in these circumstances is more legitimate. In this subsection, the different methods that apply more than a single rule in test data classification are discussed.

2.9.3.1 Confidence Group Method

Two classification methods based on utilising more than one rule's confidence to allocate the test data its class have been developed in (Thabtah, et al., 2011). The first method divides all rules contained within the test data into clusters according to the class similarity (Figure 2.8). Once this happen, the average confidence for each cluster is calculated and the test data gets allocated the class belonging to the largest average cluster's confidence. If none of the classifier rules match the test data the default is

used. The other method takes into account the classifier rules that share with the test data at least one attribute value and split them into clusters based on the class. Again the class of largest cluster confidence is allocated to the test data.

2.9.3.2 Expected Accuracy

Some AC algorithms like CPAR applied the expected Laplace accuracy for a set of rules to allocate the test data the more appropriate class. When a test data (*ts*) is requiring a class, CPAR scans over the classifier rules and saves matching rules that are contained in *ts* in a data structure. The algorithms then split the matching rules into clusters similar to confidence based classification methods and the average expected accuracy for each cluster is calculated. Lastly, *ts* is given the highest average expected accuracy cluster's class. Hereunder is the way to compute the expected accuracy for rules set cluster:

$$\text{Laplace}\,(Cluster) = \frac{(T_c(RCluster)+1)}{(T_{tot}(RCluster)+T)} \tag{2.14}$$

where

T denotes the number of class values contained in the training data set.

$T_{tot}(RCluster)$ is the total number of training examples matching a cluster's rule body .

T_c ($RCluster$) is the total number of training examples covered by a cluster's rule.

Another AC called Fitcare, (Cerf et al., 2008) has applied the average expected accuracy classification method. Experimentations using CPAR and other rule based algorithms showed that CPAR derived better quality classifiers than C4.5 with respect to accuracy on a number of UCI data sets.

2.9.3.3 Weighted Chi-Square

Chi-Square method is utilised to measure the correlation between the expected values and the observed values of a variable in a set of examples (Song, 2009) based on equation 2.17.

$$\chi^2 = \sum_{i=1}^{v} \frac{(O_i - E_i)^2}{E_i} \tag{2.15}$$

where

O_i is the observed values

E_i is the expected values.

When the expected values is different than the observed values, the correlation assumption is rejected.

CMAR algorithm was the first algorithm in AC that applied a slightly modified version of Chi-Square in 2001 to reduce the candidate rules set. In a candidate rule, CMAR checks the relationship between the rule's body and its class and discards any rule that has negative correlation based on a user defined threshold. The chi-square of *R* in CMAR is defined in equation 2.16 below.

$$Max_{\chi^2} = (\min\{Support(Antecedent), Supprt(c)\} - \frac{Support(Antecedent)Support(c)}{|T|})^2 |T| u \qquad (2.16)$$

, where

$$u = \frac{1}{Support(Antecedent)Support(c)} + \frac{1}{Support(Antecedent)(|T| - Support(c))} + \frac{1}{(|T| - Support(Antecedent))Support(c)} + \frac{1}{(|T| - Support(Antecedent))(|T| - Support(c))} + \qquad (2.17)$$

In classifying a test data CMAR groups all matching rules into clusters based on the class label. Now, when we have a single cluster (all rules having the same class) then the algorithm assigns that cluster class to the test data. Though, if more than one cluster is available, the case becomes more complicated in which CMAR measures the correlation between each cluster and the test data based on a parameter called strength. The cluster 'strength is calculated based on cluster's rules support values and their Chi-Square correlation (*Max* χ^2). CMAR assigns the test data the class that belongs to the highest cluster strength.

2.9.4 Discussion on Class Forecasting

One rule based classification for test data is considered simple approach because the largest confidence rule in the classifier matching the test data classifies it. The rule's confidence can be seen by many scholars in the AC community as the most favourable measure to distinguish among rules due to many reasons. Some of which are confidence is used for rules ranking, rules generation, and rules strength. Nevertheless, some scholars claimed that the one rule approach may not be the ideal method for classifying test data simply because there could be more than one rule contained within the test data

during classification step. These applicable rules in many cases may have close confidence values thus making only using one rule for classification doubtful.

Furthermore, for the unbalanced data sets the one rule classification approach may fail since the classifier may contain limited number of rules for the low frequency class and massive number of rules for the majority class. In this case, the majority of the test data will be classified by the majority class rules. Therefore, few scholars in AC have proposed a limited numbers of classification methods using multiple rules matching the test data to make the class assignment decision more appropriate and fair.

2.10 Chapter Summary

AC merges classification and association rule in data mining and it has recently attracted several scholars since it derives high accurate classifiers with simple chunk of knowledge. This approach comprises a number of important steps including learning the rules, rule ranking, building the classifier and classifying test data. In the last decade, a number of AC algorithms have been disseminated in the literature such as MCAR, CPAR, L^3, AC-S, uCBA and others. In this chapter, we reviewed common approaches in the literature related to each step in AC mining, and critically compared the different methods in each step. Furthermore, different data representation models such as horizontal and vertical are discussed along with their advantages and disadvantages. Next chapter we develop the first proposed algorithm in the thesis which is based on single label rules.

Chapter Three

MAC: A Multiclass Associative Classification Algorithm

3.1 Introduction

In this chapter, we investigate most of the different steps in the AC algorithm life cycle aiming to enhance them in a new AC algorithm. Specifically, we look into rule pruning, rule sorting, rule discovery and prediction steps.

One possible way to control the growth in the number of rules in AC is to develop rule pruning and sorting methods. Rule pruning can reduce rules redundancy by only keeping rules that have data coverage, and rule sorting ensures good quality rules having high rank before pruning process starts. In fact, rule sorting can be considered an early pruning step because rules that are of higher rank are usually evaluated first on the training data set and inserted into the classifier. On the other hand, rules with lower rank are often discarded during rule pruning basically because higher ranked rules have covered their training cases while constructing the classifier. This makes rule sorting a crucial step in AC since the classifier accuracy depends heavily on rules with higher ranks.

To deal with the generation of a large number of rules in AC, we propose a new pruning method which reduces the number of rules discovered without impacting the predictive accuracy of the classifiers. Furthermore, we develop a rule sorting method that limits during the ranking process a) rule random selection and b) ties among rules having similar criteria (confidence, support, length). This will have a positive effect on pruning candidate rules before producing classifiers. For rule pruning, and after

candidate rules are produced and sorted, they are considered against each training data set and the first rule that covers a training data set regardless of the class, becomes part of the classifier thereafter that training case gets removed and the same process is repeated until all training cases are discarded or all candidate rules are evaluated. This method ensures high training data coverage per rule, unlike current rule evaluation methods that require the class similarity between the training case class and the evaluated rule class which may cause overfitting on the training data. Section 3.2.3 gives further details on this method.

Furthermore, unlike most existing prediction or test classification methods in AC that uses" single rule" for making the class assignment for test data. We propose a new procedure that utilises more than one rule in making the prediction decision since there could be multiple rules in the classifier that can be applicable to the test case. Therefore, assigning the class of just a single rule may be ineffective and unfair. Our developed method takes into account all rules in the classifier matching the test data. A vote based on the number of rules is given to each group and the class belonging to the group with the largest vote is assigned to the test data. Section 3.2.4 explains the classification method thoroughly.

Moreover, the frequent ruleitem discovery step is investigated in vertical mining in order to cut down the numbers of candidate ruleitems at any given iteration. We develop a method that only intersects the TIDs of frequent ruleitems in the iteration N-1 that have the same class to come up with the candidate ruleitems in iteration N. This condition has resulted in a reduction in the number of candidate ruleitems at iteration N which enhanced this step's performance. Section 3.2.1 gives more details.

Overall, the enhancements done on each step in AC has resulted in a new algorithm named Multiclass Associative Classification (MAC) that contains

- a new pruning method in the classifier building step
- a new rule sorting method
- an enhanced frequent ruleitems discovery method in rule discovery
- a new class assignment procedure

This chapter is structured as follows: Section 3.2 illustrates MAC algorithm and its main steps such as frequent ruleitem discovery, rule extraction, rule sorting, classifier construction and test data class assignment. Section 3.3 gives a comprehensive example

to reveal MAC's insight. Section 3.4 is devoted to list the main features of MAC compared to other AC algorithms. Finally, the chapter summary is given in Section 3.5.

3.2 MAC Algorithm

The proposed algorithm utilises an AC learning strategy to generate the rules. It comprises of three main steps: rules discovery, classifier building and class assignment procedure (prediction step). In the first step, MAC iterates over the input training data set in which rules are found and devised using user specified *minsupp* and *minconf* thresholds. In the second step, the discovered rules are evaluated on training data set in order to select one subset to represent the classifier. The final step involves assigning class labels to test data. The general description of MAC is depicted in Algorithm 3.1, and details are given in the next subsections.

We assume that the input attributes are categorical or continuous. For continuous attributes any discretisation measure is applied before the training phase. Briefly and in discretising a continuous attribute, the attribute values are arranged in ascending order and the class connected with each value of that attribute is displayed. Then breaking points are identified when the value of the class changes and the gain of splitting the data for that attribute at each breaking point is computed based on IG (Quinlan, 1993). The gain of the split that represents the amount of information obtained on the class distribution within each subset of the data after the split and the split that maximises the information gained over all possible splits is chosen. The same process is repeated on the lower range of that attribute. More details can be found in (Witten and Frank, 2002). Missing attribute values will be treated as other existing values in the data set.

Input: Training data *D*, *minsupp* and *minconf* thresholds

Output: A classifier that comprises rules

1. Iterate over the training data set *D* with *n* columns to find all frequent ruleitems (Algorithm 3.2)
2. Convert any frequent ruleitem that passes *minconf* to a rule
3. Sort the rules set according to the criteria shown in Algorithm 3.3
4. Evaluate the complete set of rules discovered in step (1) on the training data set
   In order to remove rules that don't cover any training case (Algorithm 3.4)
5. Classify test data (Section3.2.4)

Algorithm 3.1 MAC algorithm

3.2.1 Rule Discovery

Most AC algorithms follow either horizontal or vertical mining approaches in the way they discover frequent ruleitems. In horizontal mining, the frequent ruleitems are found in a recursive process in which frequent 1-ruleitems discovered at iteration 1 are employed to find candidate 2-ruleitems from which the algorithm iterates over the training data to compute candidate 2-ruleitems support values to discriminate those that are frequent. At the next iteration frequent 2-ruleitems are used to generate candidate 3-ruleitems and so on for the remaining iterations.

On the other hand, in vertical mining, there is no support counting after iteration 1 since the algorithm usually stores the attribute value and the class attribute locations in the training data set in a simple array called the TID. The TID for a ruleitem is then utilised to compute the support which is simply its length. Now, when two disjoint 1-ruleitems TIDs are intersected, the resulting TID corresponds to the row numbers where the disjoint 1-ruleitems have appeared together in the training data. Then by just computing the resulting TID length (support) we can determine whether the new candidate 2-ruleitems is frequent. The same process is repeated after the second iteration and so forth.

MAC uses a training method that employs simple intersection among ruleitems' TIDs to discover the rules. The TID of a ruleitem holds the row numbers that contain the attribute values and their corresponding class labels in the training data set. The proposed algorithm discovers the frequent ruleitem of size 1 (F_1) after iterating over the training data set. Then, it intersects the TIDs of the disjoint ruleitems inF_1 to discover the candidate ruleitems of size 2, and after determining F_2, the possible remaining frequent ruleitems of size 3 are obtained from intersecting the TIDs of the disjoint ruleitems of F_2, and so forth as shown in Algorithm 3.2. The TIDs of a ruleitem comprise useful information that are utilised to locate values easily in the training data set especially in computing the support and confidence for rules. For example, for Table 3.1 and assuming support count is 3, the frequent ruleitems (size 1) ($<a_1>$, c_2) and ($<b_1>$, c_2) that are shown in Table 3.2 are used to produce the candidate ruleitem (size 2) ($<a_1, b_1>$, c_2) by simply intersecting their TIDs, i.e. (1,2,6,10) and (1,2,6) within the training data set (Tables3.1a and3.1b). The result of the above intersection is the set (1,2,6) which has a length, i.e. 3, denotes the support value of the new ruleitem ($<a_1, b_1>$, c_2). Now, since this attribute value support is larger than the *minsupp* threshold, i.e. 20%, ($<a_1, b_1>$, c_2) it

will count as frequent ruleitem. Section 3.2.5 provides a comprehensive example on the rule discovery step of MAC.

Input N-frequent ruleitems F_n

Output frequent N+1- candidate ruleitems C_{n+1}

1. $\theta \leftarrow C_{n+1}$
2. for each disjoint ruleitems (<r_i>, c), (<r_j>, c) having a common class in F_n do
3. S $\leftarrow$ TIDs ($r_{i,}$c) $\cap$ TIDs ($r_{i,}$c)
4. If |S| >= *minsupp*
5. $C_{n+1} \leftarrow$ (<$r_{i,}$ r_j>, c) $\cup$ C_{n+1}
6. end if
7. end
8. return C_{n+1}

Algorithm 3.2 MAC's frequent ruleitems discovery

This approach transforms the original training data set into a table that contains the locations (TIDs) of each attribute value and class(ruleitems) in the training data set. It employs simple intersections among these ruleitems TIDs to discover frequent ruleitems and produce the rules. Since this approach iterates over the training data set once, it is highly effective according to several experimental studies in the literature of data mining and machine learning (ML) communities especially with regards to processing time and memory usage. It should be noted that CACA and MCAR have used vertical mining for discovering the frequent ruleitems. However, unlike these rule discovery methods that consider intersecting ruleitems TIDs without having a look at the class labels our approach minimises the number of intersections considerably by only intersecting ruleitems sharing the same class. Experimental tests on the difference between our learning approach and that of these approaches is illustrated in Section3.4.

When frequent ruleitems are identified, MAC generates any of which as a rule when it passes the *minconf* threshold. Now, when an attribute value is connected with more than one class and turns out to be frequent, MAC considers only the largest frequency class associated with the attribute value and ignores the others. In cases that the class frequenciesin the training data set when connected with the attribute value is similar the choice is random.

Once the complete set of rules are derived, the sorting procedure is invoked (shown in Algorithm 3.3) to ensure that rules with high confidence are given higher priority to be

selected as part of the classifier. The rule sorting procedure has been chosen after experimentally comparing different selection criteria in Chapter 5 (Section 5.5.3) against different data sets. We have developed a rule ranking formula that minimises rule random selection. In the next sub-section, we highlight the proposed rule ranking method.

Table 3.1 Training data set

Instance number	Att1	Att2	Class
1	a_1	b_1	c_2
2	a_1	b_1	c_2
3	a_2	b_1	c_1
4	a_1	b_2	c_1
5	a_3	b_1	c_1
6	a_1	b_1	c_2
7	a_2	b_2	c_1
8	a_1	b_2	c_1
9	a_1	b_2	c_1
10	a_1	b_2	c_2

Table 3.2 Frequent ruleitems derived by MAC from Table 3.1

Frequent attribute value	**Support**	**Confidence**
<a_1>, c_2	**40%**	**57.10%**
<a_1>, c_1	30%	42.85%
<b1>, c_2	**30%**	**60%**
<b_2>, c_1	**40%**	**80%**
<a_1,b_1>,c_2	**30%**	**100%**
<a_1,b_2>, c_1	**30%**	**75%**

Table 3.1a TIDs for ruleitems belonging to Att1 of Table 3.1

a_1, c_2	a_1, c_1	a_2 , c_1	a_3 , c_1
1	4	3	5
2	8	7	
6	9		
10			

Table 3.1b TIDs for ruleitems belonging to Att2 of Table 3.1

b_1, c_2	b_2 , c_1	b_1 , c_1	b_2 , c_2
1	4	5	10
2	7		
6	8		
	9		

3.2.2 Rule Sorting

After the complete set of rules are found, rules must be sorted according to certain parameters in order to allow high predictive rules be part of the classifier that in turn is latterly used for class assignment of test data. In Chapter 2, we have surveyed ranking methods in AC and found out that the key to success is breaking ties among rules. Particularly rules that share similar criteria (confidence, support and length). To show the significance of rule ranking we use the "German" data set from the UCI data repository and apply the MCAR AC algorithm (Thabtah, et al., 2005) on this data set using 2% *minsupp* and 40% *minconf*. The number of rules derived using MCAR on the "German" data set without pruning with similar confidence values is 3443 from which 323 rules have the same confidence and support values, which make the rule preference a hard task. In fact, after pruning the classifier generated by the MCAR algorithm on this data set contained 563 rules. This simple example stresses the need for more tie breaking parameters in order to discriminate among rules during building the classifier step.

There are several different rule ranking formulas containing different criteria considered by scholars in AC. For instance, CBA algorithm (Liu, et al., 1998) and its successors consider the rule's confidence and support as main criteria for rule favouring, CMAR and MCAR algorithms add on top of that the rule's length and the majority class count respectively when rules have identical confidence and support. On the other hand, L^3G places specific rules first (rules with large number of attribute values in their body) since they claim these rules are often more accurate. Though, this approach has been criticised for ending up with very large classifiers that are hard to be maintained, understood and updated (Thabtah, 2007).

We propose that the minority class frequency as a rule preference parameter should be used when rules having similar confidence, support and length. This is since the numbers of rules for the lower frequency class are normally smaller than that of the largest frequency class. Therefore, ranking rules with smaller frequency class higher gives them a better chance to survive during rule evaluation and be part of the classifier and resulting with more class representation in the context of rules.

For the MAC algorithm, we have firstly evaluated the orders of the main different parameters used in rule ranking which are rule's confidence, support and length. Then we also tested the class distribution of class labels when two or more rules having identical (confidence, support, length). The results of the experiments clearly revealed that confidence has the highest impact on the rule ranking followed by support, rule length and lastly class frequency per rule. Though we have favoured rules associated with a less frequent class in rule ranking since such class is not well represented by rules in the classifier and usually has less number of rules.

Input: The Complete set of CARs *R*

Output: Sorted CARs *R'*

1. Given two CARs, r_a and r_b, r_a precedes r_b if :
2. The confidence of r_a is larger than that of r_b
3. The confidence values of r_a and r_b are the same, but the support of r_a is larger than that of r_b
4. The confidence and support values of r_a and r_b are the same, but r_a has less numberof attribute values in its body than r_b
5. The confidence, support and rule's body length of r_a and r_b are the same, but r_a is associated with a class that has less frequency than that of r_b inthe training data set
6. All above conditions are similar for r_a and r_b then the choice is random

Algorithm 3.3 MAC's rule sorting

3.2.3 Classifier Construction

As stated in chapter 1, AC algorithms suffer from the large number of rules generated which means that a training example is used to create many rules during the training phase. This problem necessitates developing rule pruning method. Chapter 2 categorised pruning into main categories early and late. An early pruning method for instance discards the candidate ruleitem when it fails to have enough frequency (actual support). Whereas late pruning occurs after candidate rules are produced by testing them against the training data set and removing candidate rules that do not cover any training examples. Anyhow, both types minimise the number of candidate rules and therefore the final classifier size.

So for MAC and after rules are sorted a subset of which gets chosen to comprise the classifier. Algorithm 3.4 illustrates how the classifier is built by MAC where for each training case; the algorithm iterates over the set of discovered rules (top-down fashion) and selects the first rule that matches the training case as a classifier rule. The same process is repeated until all training cases are covered or all candidate rules have been evaluated. In cases when uncovered data are left, the default class rule (the largest frequency class) will be formed from them. Finally, MAC outputs all rules that had covered training data to make the classifier. The remaining unmarked rules are discarded by the proposed algorithm since some higher ranked rules have covered their training cases during building the classifier. Therefore, these unmarked rules become redundant and useless.

The proposed rule pruning differs from other pruning procedures in AC such as CBA, CMAR, and LC in that no class similarity between the evaluated rule and the training case is required as a condition of rule significance. Rather it only considers the similarity between the rule body and the training case attributes values. This may reduce overfitting since current algorithms insert the candidate rule into the classifier if a) its body is identical to the training case values and b) it has the same class as the training case. This may result in more accurate prediction on the training data set but not necessarily on new unseen test cases. The class matching of the candidate rule and the training case does not necessarily give an additional sign of rule goodness besides the matching condition between this rule body and the training case attribute values. In other words, the performance of the classification model is not yet generalised since it has not been tested on an independent test data to measure its predictive power.

We argue that the similarity test between the candidate rule class and the training case class in building the classifier step has limited effect on the predictive power of the resulting classifiers during the prediction step. Later in Chapter 5 (Section 5.5.2), we show the main results obtained with reference to classification accuracy on different UCI data sets for both rule pruning procedures .The one which looks at the class (CBA and its successors) and the one that marks the applicable rule without checking the class(MAC). Lastly, one obvious advantage of the proposed rule evaluation method is that it ensures more data coverage per rule which consequently often leads to less number of rules in the classifier. This means the end-user can control the classifier and understand it easily.

Input: The set of generated rules (*R*) and the training data set (*D*)
Output: classifier (*Cls*)
1. *R'* = rank (*R*)
2. θ ← *Cls*
3. θ ← Temp
4. for each training case *tr* in *D* do
5. find the first rule $r_i \in R'$ that its body is contained within *tr*
6. if no rule's body is contained in *tr*
7. keep *tr* uncovered;
8. else
9. begin
10. mark r_i (take a copy of it by Temp ← Temp $\cup r_i$))
11. delete *tr*
12. end
13. end if
14. end loop
15. discard all rules in *R*
16. *Cls* ← *Cls* ∪ Temp)
17. If *D*.size > 0
18. form the majority class of the remaining unclassified cases in *D* as a default class rule
19. else
20. form the majority class rule as a default class from the current *Cls* and add it to *Cls*
21. end if

Algorithm 3.4 MAC's classifier builder

3.2.4 Classification of Test Data

When a test case is about to be classified, the prediction procedure of MAC algorithm works as follow:

It iterates over the set of the stored rules in the classifier. It highlights all rules that are contained in the test data (the rule's body matches some attribute values in the test data). If only one rule is applicable to the test data the class of that rule is assigned to the test data. In cases when multiple rules are applicable to the test data, the algorithm categorises these rules into groups according to their class and counts the number of rules in each group. The class belonging to the group that has the largest number of rules gets assigned to the test data. In case more than one group having the same number of rules, the choice will be based on the rule rank (semi-random).

This method which utilises more than one rule to make the class assignment of test data has improved test data classification procedures such as that of CBA and its successors that take the class of the first ranked rule in the classifier matching the test data to make the prediction decision. Furthermore, few multiple rule prediction methods in AC like CMAR employs mathematical based attribute assessment formulas, e.g., weighted Chi-Square and require extensive computations especially when there are many rules. Our method is simple yet effective in predicting the class of test data in the context of classification accuracy as we will see in Chapter 5. Lastly, in cases when no rules in the classifier are applicable to the test case the default class rule will be assigned to that case.

3.3 Example on MAC

In this section, we demonstrate a comprehensive example to simplify the MAC algorithm steps for the reader. Table 3.3excluding the last column on the right hand side displays a sample training data set. Assume that *minsupp* and *minconf* have been set to 20% and 75% respectively for presentation purpose.

3.3.1Frequent Ruleitems Discovery and Rule Generation

After scanning the data, frequent ruleitems of size "1" are shown in Table 3.4 along with their TIDs and support count. Candidate ruleitems ($<a_3>,c_1$), ($<b_2>,c_2$), ($<d_1>,c_1$)have been discarded as they did not have enough support values (their frequencies are below the *minsupp* threshold) and this is the reason why they are not in Table 3.4.

Once the frequent 1-ruleitems are discovered, MAC uses their TIDs to find out the candidate 2-ruleitems. In particular, the TIDs of ruleitems belonging to different attributes and having similar class labels are intersected in order to determine whether the new formed 2-ruleitems are frequent. Unlike other current vertical algorithms that intersect the TIDs of disjoint ruleitems without looking at their class labels in order to

find frequent ruleitems, our algorithm intersects the TIDs of two ruleitems if they belong to the same class which surely reduces the numbers of intersections. For example, the number of TIDs intersections performed by MAC on the frequent 1-ruleitems of Table 3.4 to discover the candidate 2-ruleitems is 17. On the other hand, other vertical algorithms like MCAR require 33 TIDs intersections on the same frequent 1-ruleitems which means our algorithm cuts down the number of TIDs intersected at iteration (1) by almost 95%. Furthermore, the minimisation in the number of TIDs

Table 3.3Sample data

Instance number	Att_1	Att_2	Att_3	Class		**Rule applied in pruning**
1	a_1	b_1	d_1	c_2		**R1**
2	a_1	b_1	d_2	c_2		**R1**
3	a_2	b_1	d_2	c_1		**R3**
4	a_1	b_2	d_2	c_1		**R7**
5	a_3	b_1	d_1	c_1		**R1**
6	a_1	b_1	d_1	c_2		**R1**
7	a_2	b_2	d_3	c_1		**R2**
8	a_1	b_2	d_3	c_1		**R2**
9	a_1	b_2	d_2	c_1		**R7**
10	a_1	b_2	d_2	c_2		**R7**

Table 3.4 Frequent 1-ruleitems produced from the data in Table 3.3

1-Ruleitem		Support count	TIDS	Confidence
attribute	**Class**			
ATT (1)				
a_1	c_2	4	(1,2,6,10)	57.14%
a_2	**c_1**	**2**	**(3,7)**	**100.00%**
a_1	c_1	3	(4,8,9)	42.85%
ATT (2)				
b_1	c_2	3	(1,2,6)	60.00%
b_1	c_1	2	(3,5)	40.00%
b_2	**c_1**	**4**	**(4,7,8,9)**	**80.00%**
ATT (3)				
d_1	c_2	2	(1,6)	66.67%
d_2	c_2	2	(2,10)	40.00%
d_2	c_1	3	(3,4,9)	60.00%
d_3	**c_1**	**2**	**(7,8)**	**100.00 %**

Table 3.5 Frequent 2-ruleitems produced from the data in Table 3.4

1-Ruleitem		Support count	TIDS	Confidence
attribute	**Class**			
(a_1, b_1)	**c_2**	**3**	**(1,2,6,10) ∩ (1,2,6) = (1,2,6)**	**100.00 %**
(a_1, b_2)	**c_1**	**3**	**(4,8,9) ∩ (4,7,8,9) = (4,8,9)**	**75.00%**
(a_1, d_1)	**c_2**	**2**	**(1,2,6,10) ∩ (1,6) = (1,6)**	**100.00 %**
(a_1, d_2)	c_2	2	(1,2,6,10) ∩ (2,10) = (2,10)	50.00%
(a_1, d_2)	c_1	2	(4,8,9) ∩ (3,4,9) = (4,9)	50.00%
(b_1, d_1)	c_2	2	(1,2,6) ∩ (1,6) = (1,6)	66.67%
(b_2, d_2)	c_1	2	(4,7,8,9) ∩ (3,4,9) = (4,9)	66.67%
(b_2, d_3)	**c_1**	**2**	**(4,7,8,9) ∩ (7,8) = (7,8)**	**100.00 %**

intersections of N-ruleitems results in smaller number of candidate N+1-ruleitems by the proposed algorithm.

When the frequent 1-ruleitems are discovered, MAC utilises their TIDs to derive candidate 2-ruleitems from which then frequent 2-ruleitems are identified and shown in Table 3.5. It should be noted that there were 17 different candidates 2-ruleitems generated from which only 8 have turned to be frequent. The algorithm continues the process of generating candidate 3-ruleitems from frequent 2-ruleitems (Table 3.5) and determines frequent 3-ruleitems as depicted in Table 3.6. There were two candidates' 3-ruleitems that did not pass the *minsupp* requirement and thus were deleted.

Once we reached frequent 3-rulitems, the algorithm stops processing frequent ruleitems and starts evaluating the confidence values for the complete set of frequent ruleitems found so far and produces those which pass *minconf* threshold as class association rules (CARs). Therefore, from (Tables 3.4-3.6) the CARs or rules for short derived by the MAC are in bold within these tables. All other rules are removed by the algorithm and thus MAC has discovered only eight candidate rules from Table 3.3.

3.3.2 Classifier Construction

Once the rules are extracted, they get sorted according to the four parameters associated with them (confidence, support, rule's length and minority class frequency). The rules after sorting are displayed in Table 3.7. In rule evaluation (pruning), and for each training case shown in Table 3.3excluding the last column, MAC iterates over the rules (top down) fashion and chooses the rule contained within the training case. So, for the first and the second training cases, rule #1 covers them. The third training case is covered by rule #3 and the fourth training case is covered by rule #7 as shown in Table3.7.

Rule #2 covers training cases seven and eight, and training case six is covered by rule #1. The last two training cases nine and tenare covered by rule #7. There is one training case that no candidate rule was able to cover which is case five. For this case, MAC's rule evaluation method takes on the highest sorted rule that is partly applicable to this training case which is rule#1, instead of taking on the default class rule. The rules in italic within Table 3.7 represent the MAC classifier which contains four rules in this example. The remaining rules which are neither bold nor italic represent redundant rules that were unable to cover any training data and therefore are discarded.

3.3.3 Class Assignment

Once the classifier is formed by MAC, the classification process becomes ready. So assume that the test data is given in Table 3.8, MAC applies the classifier on the test data as follows:

For the first test data, the classifier rules (bold rows in Table 3.7) that are applicable to it are (#2,#3,#7) where all these rules are associated with class "$\mathbf{c_1}$" so this test data is assigned class "$\mathbf{c_1}$". For the second test data, it will be assigned class "$\mathbf{c_2}$" since only classifier rule #1 is applicable to it. For the third and the fourth test data, there is one rule in the classifier that can be applied on them (#7) so it will be fired. Finally, the last test data is assigned either class "$\mathbf{c_1}$" or "$\mathbf{c_2}$" since two rules (#1, #7) are relevant to it. The choice in this case is random but MAC prefers the rule with the higher rank which is #1.

Table 3.6 Frequent 3-ruleitems produced from the data in Table 3.5

1-Ruleitem		Support count	TIDS	Confidence
attribute	**Class**			
(a_1, b_2 , d_2)	c_1	2	(4,8,9) ∩ (4,9)) = (4,9)	66.67%
(a_1, b_1 , d_1)	**c_2**	**2**	**(1,6) ∩ (1,6) = (1,6)**	**100.00 %**

Table 3.7 Sorted candidate rules produced by MAC

Rulerank(Algorithm 3.3)	**Rule**		**Support**	**Confidence**
1	***(a_1, b_1) → c_2***	***c_2***	***3***	***100.00 %***
2	***d_3 → c_1***	***c_1***	***2***	***100.00 %***
3	***a_2 → c_1***	***c_1***	***2***	***100.00%***
4	(a_1, d_1) → c_2	c_2	2	100.00 %
5	(b_2, d_3) → c_1	c_1	2	100.00 %
6	(a_1, b_1 , d_1) → c_2	c_2	2	100.00 %
7	***b_2 → c_1***	***c_1***	***4***	***80.00%***
8	(a_1, b_2) → c_1	c_1	3	75.00%
	Default rule	***c_1***		

Table 3.8 Test data

Instance number	**Att1**	**Att2**	**Att3**	**Predicted Class**	**Rule(s) applied**
1	a_2	b_2	d_3	c_1	(#2,#3,#7)
2	a_1	b_1	d_1	c_2	#1
3	a_1	b_2	d_2	c_2	#7
4	a_1	b_2	d_2	c_1	#7
5	a_1	b_1	d_3	c1	#1

3.4 MAC vs Other AC Algorithms

In the research literature of AC mining, there are several different algorithms developed most of which are CBA, lazy (L^3), MCAR or CMAR based. In other words, the majority of current AC algorithms are developed to enhance either the predicative power or the

efficiency of the abovementioned algorithms. The main distinctions between the proposed algorithm and these algorithms are the following:

- The methodology of forming the classifier from the discovered set of rules in the proposed algorithm does not consider the similarity between the candidate rule class and that of the training case during the selection of the classifier rules. This reduces the final classifier size because the rule in this case normally covers more training data. On the other hand, other AC algorithms insert the rule into the classifier particularly if the candidate rule class is identical to that of the training case.
- The class assignment procedure of the proposed algorithm is a voting method that takes into consideration multiple rules in predicting the class of test data instead of just one rule as most other AC algorithms. Further, this method differs from that of CMAR since it is based on counting rules and does not require computing weighted chi-square per candidate rule.
- The ranking process of MAC focuses on breaking more ties and prefers minority class when many rules having similar criteria. This had minimised arbitrary rule selection in ranking process.
- MAC utilises a modified TIDs intersection among frequent ruleitems that minimises the number of intersections whereas the majority of current AC algorithms employ level-wise or greedy searches in discovering the rules (Apriori, CMAR or CPAR approaches). There are few algorithms like CACA and MCAR that use a vertical learning approach adopted from (Zaki and Gouda, 2003) though MAC learning method intersects only ruleitems sharing the same class labels at each step. This reduces the number of candidate ruleitems at each iteration during the process of finding frequent ruleitems.

3.5 Chapter Summary

In this chapter, a novel AC algorithm called MAC that employs a new methodology in the construction of rules has been proposed. This resulted in saving too many useless TIDs intersections and therefore the number of candidate ruleitems at a given iteration has been minimised. Moreover, MAC employs a class prediction method based on group voting to ensure that all rules in the classifier that are applicable to the test case participate in the class assignment for that test case. Finally, we have investigated the rule sorting step to select the most effective criteria to rank rules before building the classifier. This has ended up with a new rule ranking method. Experimental results on

MAC are discussed in Chapter 5. In the next chapter, we extend MAC algorithm to produce rules with multiple class label from data sets which results in a new type of knowledge.

Chapter Four

MCAC: A Multi-label Classifier based Associative Classification

4.1 Introduction

Constructing multi label rules from data examples that are connected with one class in AC is considered one of the challenging problems that have recently attracted scholars because of its applicability in real world applications. Still the number of multi-label rules methods is rare since the majority of the current AC methods consider generating rules connected to one class and do not pay attention to other classes linked with them in the training data set. We have demonstrated in Chapter 1 an example that signifies the importance of extracting all class labels per rule from single label data sets.

Let's revisit the medical diagnosis example in Chapter 1 and assume symptoms such as nausea, headache and sore throat could relate to three types of illness "tonsillitis", "migraine" or "flu", which are stored in a database. Assume also that the frequencies of the symptoms (nausea, headache, sore throat) together in the database are38, 36 and 26with "tonsillitis", "migraine" or "flu" classes, respectively. Now, an AC algorithm discovers only the rule associated with class ("tonsillitis"), and does not consider the other existing classes with attribute values (nausea, headache, sore throat). Though, it is beneficial to find the other rules since they represent valuable information having a sufficient representation in the database. In this chapter, we handle the generation of multi-label rules from single label classification problems.

To achieve our goal, this chapter proposes a new multi-label rules algorithm based on AC called "Multi-label Classifier based Associative Classification" (MCAC). This algorithm extracts from data sets not only rules with the most obvious class but rules that

are associated with a ranked set of classes. When an attribute value is connected in different rows within the training data sets with more than one class, the proposed algorithm extracts all these classes. Thus, later in the prediction step, there can be more alternatives (classes)when the rule is fired particularly in assigning the appropriate class(s) for an incoming test case.

The MCAC algorithm generates multi-label rules from the whole training data set and without training on separate parts. In addition, a procedure that merges both "single rule" and "group of rules" test data classification approaches is proposed to take the advantages of both approaches. Chapters 5- Section 5.6.2, and chapter 6 demonstrate the applicability of MCAC on real data related to two important applications (the trainer scheduling for a financial institution (Chakhlevitch and Cowling, 2008)) and (website phishing classification (Mohammad, et al., 2012))besides over 20 UCI data sets (Merz and Murphy, 1996).

The proposed algorithm including data representation, rule discovery, constructing the classifier, and class assignment are discussed in Section 4.2.A comprehensive example on MCAC is presented in Section 4.3, and lastly MCAC's features and the chapter summary are given in Sections 4.4 and4.5 respectively.

4.2 The MCAC Algorithm

In this section, we elaborate on the way that MCAC finds the rules, makes the classifier, and predicts test data. Figure 4.1 illustrates the general life cycle that our algorithm must go through in which data is assumed in the figure to be processed. The first major phase is the mining for rules which consists of two sub-steps: frequent ruleitems discovery rules extraction. In this phase, the algorithm also merges any of the resulting rules that have the same antecedent and are linked with different classes to produce the multi-label rules. Further, the class probabilities computation and ranking of the classes in the multi-label rules is also performed.

The second major phase is about making the classifier and involves mainly sorting the rules and removing useless ones. The outcome of the second phase is the classifier which contains single and multi-label rules. The final major phase concerns about predicting test data instances. It involves testing the classifier on an independent data set to measure its

performance. All these phases are fully explained in MCAC in the next sub-sections, and the general description of MCAC algorithm steps is displayed in Algorithm 4.1.

The proposed algorithm assumes that the input attributes in the training data set are categorical (having distinct values), and for each of these attributes, all possible values are mapped to a set of positive integers. For continuous attributes any discretisation method can be employed.

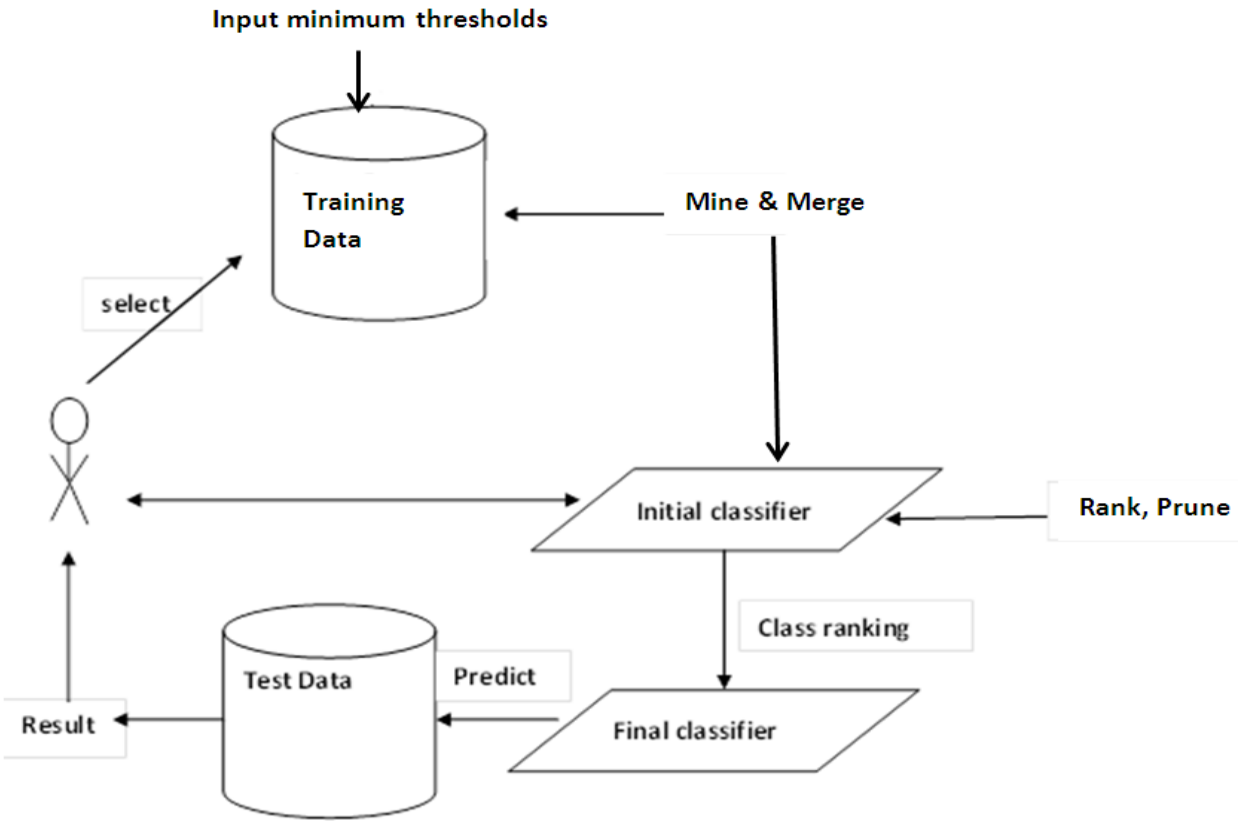

Fig. 4.1: The life cycle for MCAC's algorithm

4.2.1 Data Representation

As mentioned in Chapter 3, there are two main data layouts in AC adopted from association rules literature. These are horizontal and vertical (Zaki and Gouda, 2003). The majority of AC algorithms have utilised the horizontal data format to represent the input data where each training case is linked with a unique number followed by a list of attribute values inside the case. On the other hand, a training data set in the vertical data layout consists of a

group of attribute values, where each attribute value is followed by its locations in the training data set. A number of research studies (Thabtah, 2007; Zaki and Gouda, 2003) revealed that the vertical data format is more effective for representing data than the horizontal format since it makes the process of identifying frequent attribute values efficient, specifically the task involving the support counting. This is simply because vertical algorithms use TIDs intersection among attribute values to accomplish the task of support counting. In the proposed algorithm, the vertical data format is used to represent the training data set before frequent attribute discovery and rule generation processes begin.

Input: Training data *D*, minimum confidence (*minconf*) and minimum support (*minsupp*) thresholds
Output: A classifier

1. Preprocessing: Discretise continuous attributes if any
2. can the training data set *T* to discover the complete set of frequent attribute values
3. Convert any frequent attribute value that passes *minconf* to a single label rule
4. Form any two or more single label rules that have identical body and different class to derive the multi-label rules
5. Compute the support and confidence of the multi-label rules and assign probabilities
6. Sort the rule set according to certain conditions (Section 4.2.3)
7. Rank the class labels in the multi-label rules based on their probabilities
8. Apply the classifier building procedure (Section 4.2.3) to produce the classifier (Cm)
9. Form the default class rule
10. Classify test data using rules in Cm (Figure 4.3)

Algorithm 4.1 General steps performed in MCAC's phases

4.2.2 Rules Discovery

In this section, we describe the process of finding and generating the multi-label rules. Hereunder are the main sub-steps in this phase:

1) The generation of frequent ruleitems. These are the attribute values plus class in the training data set that have passed the *minsupp* requirement and can participate in the rules generation.

2) The generation of the single label rules. These are normally produced from the frequent ruleitems and have passed the *minconf* requirement.

3) The production of the multi-label rules: these are derived by merging two or more single label rules sharing the same antecedent but having different classes.

4) Compute the class probability per rule.

In this section, we discuss in details each of these sub-steps.

Similar to MAC algorithm, MCAC uses a rule discovery method that utilises an intersection among ruleitems TIDs to discover the rules. The learning method of the proposed algorithm discovers the frequent ruleitems of size 1 (F_1) after scanning the training data set once. In particular, for each attribute value linked with a class, its support is computed from its TIDs. Only the largest TID (most frequency class) of an attribute value denotes the attribute value support count. In cases where an attribute value is connected with more than one class in different places in the training data set it will end up with more than one support count. This process ensures the discovery of the multiple label rules since the algorithm allows an attribute value to have more than one class as long as they are frequent (passed the *minsupp* requirement).

When F_1 is generated, the algorithm simply intersects the TIDs of the disjoint ruleitems inF_1 to discover the candidate ruleitems of size 2.After determining F_2, the possible remaining frequent ruleitems of size 3 are obtained from intersecting the TIDs of the disjoint attribute values of F_2, and so on. It is worth to note that the proposed algorithm considers only disjoint attribute values in F_n having the same class labels in generating C_{n+1}.This definitely saves many unnecessary intersections among ruleitems during this step. As mentioned in Chapter 3, the TIDs of ruleitems contain useful information that are utilised to locate attribute values and class values easily in the training data set especially in computing the support and confidence for rules.

For example, assume that support count is set to 4, the frequent ruleitems (size 1) of Table 4.1 are (<y>, cl_1) and (<x>, cl_3), (<a>, cl_3) , and (<a>, cl_1).These are the input for producing the candidate 2-ruleitems by simply intersecting their TID within the training data set. So to determine whether the new 2-ruleitems(<x,a>, cl_3) is frequent we intersect the TIDs of ruleitems(<x>, cl_3) and (<a>, cl_3), e.g. (3,5,11,12) and (4,6,7,11) respectively. The result of the above intersection is the TID (11) which its cardinality, e.g.1, denotes the support of the new attribute value (<x,a>, cl_3). This ruleitem is discarded since its support is smaller than the *minsupp* threshold, i.e. 4/15.

This rule discovery method iterates over the training data set once similar to MAC algorithm in finding the rules. One primary difference between our frequent ruleitems discovery algorithm and the few abovementioned ones in Chapter 2 is that we intersect only

disjoint attribute values having identical class at any given iteration which reduces costs associated with the numbers of intersections.

When frequent attribute values plus their classes (ruleitems) are identified, MCAC generates any of which as a rule when it passes the *minconf* threshold. This is accomplished in a direct manner since all necessary information for calculating the rule's confidence values are stored in the ruleitems TIDs. Any frequent ruleitem that holds confidence smaller than the *minconf* is discarded.

We recall that the proposed algorithm keeps all class labels associated with an attribute value of any length during the rule discovery as long as the attribute value and these class labels together survive the *minsupp* threshold. This is unlike other typical AC which only keeps one class connected with the attribute value in the training data (often the one that occurs the most with the attribute value). In cases when two or more class labels are associated with an attribute value, these algorithms discard them and only keep the most frequent class.

Table 4.1 Sample data

$Attribute_1$	$Attribute_2$	$Attribute_3$	Class
y	z	b	cl_1
y	z	b	cl_2
x	a	b	cl_1
x	a	b	cl_3
y	a	d	cl_1
x	a	d	cl_3
y	a	b	cl_3
y	a	b	cl_1
y	a	d	cl_1
y	a	d	cl_2
y	a	d	cl_3
x	g	b	cl_3
x	g	d	cl_3
x	g	d	cl_1
x	g	b	cl_2
x	g	b	cl_3

For any attribute value connected with two or more classes that is frequent, MCAC algorithm generates multi-label rules for them if they pass the *minconf* threshold. For example, the attribute value <a> of Table 4.1 is linked with two classes, e.g. (cl_1, cl_3) 4 times each in the training data set. Since *minsupp* and *minconf* are set to 4/15 and 40% respectively, ruleitems (<a>, cl_1), and (<a>, cl_3) have higher support and confidence than the *minsupp* and *minconf* thresholds. Thus, two rules can be produced in this case: $a \rightarrow cl_3$, and $a \rightarrow cl_1$. For this example, a typical AC algorithm such as CBA or CMAR only derives

the rule that has higher coverage in the training data. Meaning any of the above single label rules are generated because they have similar count. On the other hand, the proposed algorithm does not discard any useful knowledge and for the above ruleitems. It produces a multi-label rule *R*: $a \rightarrow cl_1 \vee cl_3$, where normally classes are ranked based on their count with the attribute values.

Since MCAC algorithm devises rules with multiple label, there should be a way to rank theseclasses in the rule's consequent since the number of occurrences for each class with the rule's body (attribute values) may differ. We need to ensure giving classes with larger number of occurrences higher rank within the rule so the classifier can benefit later during the class assignment in regards to predictive performance.In the proposed algorithm and for a multi-label rule (*r*), class cl_a precedes class cl_b ($cl_a \prec cl_b$) if the frequency of $cl_a$with r's attribute values is greater than that of cl_b in the training data set. Our algorithm assigns a weight or a probability for each class in the multi-label rule (*r)* that denotes $\frac{|r_c|}{\left|\sum_{i=1}^{n} r_{c_i}\right|}$ (4.1).

For the multi-label rule (r): $a \rightarrow cl_1 \vee cl_3$ mentioned earlier, the probabilities assigned by the MCAC algorithm to classes cl_1 and cl_3 are 4/8 and 4/8 respectively. These probabilities are laterally used instead of the rule's classes in forecasting test data in the prediction step. So when a test data (a) is about to be classified and rule r is found to be applicable MCAC assigns 0.5 to a. In other words, the true rule's class probability is given to the test data rather the class itself.

4.2.3 Classifier Construction

Once the complete set of rules is derived, a rule sorting procedure is invoked to ensure that rules with high confidence and support values are given higher priority to be selected during building the classifier (Rule evaluation). The rule sorting procedure utilised considers different criteria to favour among rules. The criteria order is: rule's confidence, support, length and class frequency (least frequent or minority class).

For the multi-label rules, there will be new confidence and support values assigned to them, and specifically for each multi-label rule, the average confidence and support for its single label rules are assigned to it. We recall that the class labels that are connected with

the multi-label rule have passed the *minsupp* and *minconf* requirement during the rule discovery phase.

MCAC computes the average support and confidence for all single label rules that form the new multi-label rule and assign them to the multi-label rule. This ensures the right position of the multi-label rule in the set of candidate rules. Our approach gives the new multi-label rule the appropriate rank that may increase its chance to be part of the classifier and thus latterly utilised in predicting test data. Once the multi-label rule is generated, MCAC allocates each of its class a probability that denotes the real support count of it when linked with the rule's body in the training data set. This probability will be assigned to test data when the rule is used in the classification step.

After rules are sorted, a subset of these rule are then chosen to build the classifier. Precisely, and for each training case, MCAC goes over the complete set of candidate rules and inserts any rule that is applicable to a training case into the classifier and removes all data relevant to the rule (positive instance). The rule data coverage does not necessitate the similarity between the rule's class and that of the training case. The same process is repeated until all training cases are removed(covered) or all candidate rules have been tested. Finally, the algorithm derives the classifier which is basically all candidate rules that have covered training cases. Any remaining rules that have no data coverage are discarded. In cases when there are unclassified cases remaining in the training data set, a default rule for the largest count class linked with the unclassified cases is formed.

4.2.4 Class Assignment Procedure

MCAC fires the first sorted rule in the classifier applicable to the test case and assigns its probability to the test case as displayed in Algorithm 4.2. The rule attribute values must be identical to the test case in order to be chosen for classifying the test case class. When there is no rule fully applicable to the test case then we take on the group of rules that contained in the test case attribute value, and divide them into groups similar to the MAC prediction procedure (Chapter 3). We then count the number of rules per group and assign the test case the class that belongs to the group with the maximum number of rules. In cases when there are two groups with the maximum number of rules the choice is random.

This prediction procedure ensures that only rules with high quality in the classifier are used to forecast test data. In addition, unlike the majority of current prediction procedures

that take on the default class when no rules are found applicable to the test case, our procedure minimises the utilisation of the default rule in this step which normally improves upon the resulting classifier accuracy. This is since default rule has been created with high error from the remaining unclassified training data cases while building the classifier and reducing its usage in the prediction step is an advantage that normally results in an improvement.

```
Input: test data set (Ts), Classifier (Cls)
Output: Classification Accuracy Cy

1.  For each test case in Ts Do
2. For each rule r in Cls Do
3.        If there exists a rule r that fully contained in ts
4.            assign r's class to ts
5.        else
6.   begin
7.        find all partly rules of Cls in ts
8.        group the relevant rules per class
9.        count the number of rules per group
10.       assign the class that belongs to the maximum number of rules to ts
11. end
12.         end if
13.     else assign the default class to ts
14.    end if
15. end
16. end
17. compute the total number of errors of Ts
```

Algorithm 4.2 Class assignment procedure of MCAC algorithm

4.3 MCAC Example

We show in this section an example on how MCAC generates and produces the multi-label rules and how class labels belonging to a rule are sorted in its consequent. We also highlight the novelty of the rule discovery phase in which we demonestrate how hidden knowledge in the training data set are derived by our algorithm and another algorithm called MMAC. We want to show that MCAC is more practical learning algorithm which

devises additional hidden knowledge that may benefit the performance of the classifier and the end user.

Assume that *minsupp* and *minconf*have been set to 20% and 40% respectively. Table 4.2 displays an initial training data set, and the candidate rules extracted are depicted in Tables 4.3a.While the algorithm is generating the rules, it checks whether there exists a candidate rule that is already extracted with a similar body of the current rule. If this condition is true, then the algorithm appends the current rule with the already extracted rule to form a new multi-label rule. For example, in Table 4.3a, the attribute value $<a_1>$ is connected with two class labels, i.e. (c_2,c_1), with frequencies 4 and 3 respectively. Current AC algorithms will produce only one rule for this attribute value, i.e. $a_1 \rightarrow c_2$ and simply discards class (c_1) because (c_1) has more numberof occurences in the training data set with attribute value $<a_1>$. However, MCAC produces a multi-label rule for $<a_1>$ as $a_1 \rightarrow c_2 \vee c_1$. The same scenario appliesto attribute value $< b_1>$. These additional knowledge are vital for many reasons:

Table 4.2 Initial training data set

Instance number	Att_1	Att_2	Class
1	a_1	b_1	c_2
2	a_1	b_1	c_2
3	a_2	b_1	c_1
4	a_1	b_2	c_1
5	a_3	b_1	c_1
6	a_1	b_1	c_2
7	a_4	b_2	c_1
8	a_1	b_2	c_1
9	a_1	b_3	c_1
10	a_1	b_2	c_2

1) The decision makers now have more than one solution for a possible scenario where they can benefit from the additional knowledge in making decisions

2) The predictive accuracy may improve since multiple classes are associated with a rule with different possible probabilities based on their frequencies in the training data set. So when a test data is about to be classified, there will be more than one option to assign the class so we can end up with partial classificaiton (partial hit) rather wrong classification (using one class approach).

3) Less possible candidate rules to be evaluated during the formation of the classifier, consequently minimising the effort in building classifier.

The candidate multi-label rules must pass the *minsupp* and *minconf* in order to be considered part of the classifier and their actual support and confidence values are updated when they are formed as displayed in Table 4.3a. The rules in bold within Table 4.3 represent the possible candidate multi-label rules shown in Table 4.3a. Once the rule extraction is finished,MCAC sorts all possible candidate rules according to confidence, support, rule length and class distribution frequency. The candidate rules are ready for evaluation against the training data set in order to choose the best ones that can make the classifier.Ranking of classes within the multi-label rules is performed based on each class frequency as explained in Section 4.3.2.

So once all candidate rules are extracted and sorted by the MCAC algorithm, they will be tested on the training data to form the classifier. Table 4.4 shows the classifier devised by our algorithm that consists of four rules, two of which are multi-label ones.

Let's look at MMAC algorithm behaviour of rule generation and classifier formation because MMAC and MCAC generate the same form of output (disjunctive multi-label rules)for fair comparison. So we compare the MMAC ways ofgenerating the rules and forming the classifier withthose of our algorithm on the data displayed in Table 4.2. The candidate rules set derived by the MMAC algorithm is shown in Table 4.5 in which rule ranking based on confidence, support and rule length is applied. The MMAC algorithm considers a rule part of the classifier when it covers a training case and has the same class of the that case. So, only three candidate rules have training data coverage and are shown in

Table 4.3 Candidate rules produced from the data in Table 4.2

Ruleitems		Support count	Confidence
attribute	**Class**		
a_1	**c_2**	**40%**	**57%**
a_1	**c_1**	**30%**	**42%**
b_1	**c_2**	**30%**	**60%**
b_1	**c_1**	**20%**	**40%**
b_2	c_1	30%	75%
$a_1 \wedge b_1$	c_2	30%	100%
$a_1 \wedge b_2$	c_1	20%	66%

Table 4.3a Candidate multi-label rules produced from the data in Table 4.2 via MCAC

Ruleitems		Support	Confidence
attribute	**Class**		
a_1	c_2,c_1	35%	50.00%
b_1	c_2, c_1	25%	50.00%

Table 4.4 The classifier of MCAC algorithm from the data in Table 4.2

	Ruleitems		Support count	Confidence
RuleId	attribute	**Class**		
R1	$a_1 \wedge b_1$	c_2	30%	100%
R2	b_2	c_1	30%	75%
R3	a_1	c_2,c_1	35%	50.00%
R4	b_1	c_2, c_1	25%	50.00%

bold withinTable 4.5and the remaining rules aredeleted. The unclassified instances of Table 4.2 (rows 3,5,9) will form a new training data set as displayed in Table 4.6.

The MMAC continues extracting the rules from the remaining uncovered cases in Table 4.6 where only one rule is devised which is $b_1 \rightarrow c_1$. When this rule is evaluated against Table 4.6, it covers two cases and therefore it will be added to the classifier and only one training case is left unclassified which will form the default class rule (c_1). The classifier produced by the MMAC algorithm on Table 4.2 is depicted in Table 4.7 in which it consists of four rules none of which is multi-label and a default class rule. The fact that the proposed algorithm,MCAC, was able to discover 2 additional multi-label rules from

Table 4.5 MMAC candidate rules produced from the data in Table 4.2

	Ruleitems		Support count	Confidence
RuleId	attribute	**Class**		
R1	$\mathbf{a_1 \wedge b_1}$	$\mathbf{c_2}$	**30%**	**100.00%**
R2	$\mathbf{b_2}$	$\mathbf{c_1}$	**30%**	**75.00%**
R3	b_1	c_2	30%	60.00%
R4	$\mathbf{a_1}$	$\mathbf{c_2}$	**40%**	**57.14%**

Table 4.6 Training data *T'*- iteration 2 for uncovered data- MMAC

Instance number	Att_1	Att_2	Class
3	a_2	b_1	c_1
5	a_3	b_1	c_1
9	a_1	b_3	c_1

Table 4.7 MMAC classifier derived from the data in Table 4.2

Classifier Rule		Support count	Confidence
attribute	**Class**		
$a_1 \wedge b_1$	c_2	30%	100%
b_2	c_1	30%	75%
a_1	c_2	30%	60.00%
b_1	c_1	20%	40%

limited number of examples which MMAC algorithm was unable to find is a practical evidence on the novelty of the rule discovery and the generation phase of our algorithm.

To have an insight look on the behaviour of both MCAC and MMAC during making the classifier, Table 4.8 depicts the candidate rules applied by both algorithms. The table clearly shows that MCAC has covered completetly the training data set at once and four candidate rules have made it to the classifier level. On the other hand, MMAC rules had covered seven out of ten training examples and the algorithm is forced to create a new training data consisting of the remaining three examples to discover one additional rule. Actually, MMAC has to learn twice from two parts of the training data set to generate the classifier whereas our algorithm extracted the classifier at once yet finding additional knowledge missed by MMAC.

Table 4.8 Classifier Building (MMAC vs MCAC)

Example number	Att_1	Att_2	Class	MMAC	MCAC
1	a_1	b_1	c_2	R1	R1
2	a_1	b_1	c_2	R1	R1
3	a_2	b_1	c_1		R4
4	a_1	b_2	c_1	R2	R2
5	a_3	b_1	c_1		R4
6	a_1	b_1	c_2	R1	R1
7	a_4	b_2	c_1	R2	R2
8	a_1	b_2	c_1	R2	R2
9	a_1	b_3	c_1		R3
10	a_1	b_2	c_2	R4	R2

4.4 MCAC vs Other AC Algorithms

The role of this section is to shed the light on the main characteristics of the proposed algorithm especially in the context of multi-label rules AC mining. We have mentioned earlier that there are limited numbers of multi-label rules AC algorithms such as lazy CLAC and MMAC. The main distinctions between the proposed algorithm and these algorithms are the following:

- MCAC produces multi-label rules during the process of learning (training phase) without the need to perform recursive learning step (repetitive steps)as MMAC. MCAC has derived the classifier shown in Table 4.5while MMAC is still reiterating the uncovered data to find more potential rules. The recursive learning step in the MMAC algorithm

necessitates mining different parts of the training data many times, and consequently this requires an independent step to generate the multi-label rules.

- The MMAC algorithm produces N single label classifiers locally since it mines parts of the training data rather the complete data set once whereas MCAC algorithm produces a global classifier at once. Also we would like to highlight that the uncovered data by MMAC in iteration 1 were actually classified by MCAC's multi-label classifier (Table 4.8). Instance 3 and 5 were assigned rule R4 and instance 9 was assigned rule R3, both being multi-label rules classifiers.
- The prediction of test data in the proposed algorithm involves employing a procedure that takes into account identical similarity between the rule body and the test data attributes value. When this evaluation fails our prediction procedure takes on the group of rulesin the classifier that partly contained in the test data for class assignment. This is unlike other current algorithms which assign the test data immediately the default rule class which usually leads to high chance of error.
- The Lazy CLAC as well as Rank-label delays the rule inducing process until the classification phase, specifically when the test data is about to classify. In other words, there is no global classifier learnt in CLAC rather it creates local classifiers on demand when a test data requires class assignment. This may causes too many data projections between the training and test data when a test data is about to be classified and a local classifier for each test data. On the other hand, MCAC algorithm generates the multi-label global classifier only once and uses rules within the classifier to classify test data.

4.5 Chapter Summary

In this chapter, a new multi-label rules algorithm based on AC mining called MCAC has been proposed. MCAC extracted multi-label rules from the complete training data set discovering all classes connected with a rule. The rule in the MCAC classifier may contain more than one class in its consequent in a disjunctive manner where each class is associated with a weight/probability. The MCAC algorithm consists of three main phases: rule learning, classifier construction and classification. The novelty of the proposed algorithm can be summarised hereunder:

The general distinguishing features of the proposed algorithm can be summarised hereunder:

1) The ability to find and extract rules having multiple class labels from single label data sets so the end-user can end up with highly rich knowledge base (classifiers) that can be explored any time in decision making.
2) The learning mechanism employed by MCAC enables discovering a ranked class set per rule that is associated with probabilities computed from the training data set.
3) When a rule is fired in the classification step, the probabilities connected to the rule's classes are allocated to the test data rather than the classes themselves. This ensures better classifier's accuracy since probabilities between 0 to 1 are assigned to test data which consequently reduces the number of misclassifications.
4) The prediction procedure of MCAC reduces the use of default class because it allows a hybrid method that allows group based partial rule matching when no identical rule can be found.

In the next chapter, we show the implementation and evaluation of MCAC and MAC algorithms on different data sets related to UCI, and real application. Details about MCAC experimentations and results analysis are discussed in Chapter 5.

Chapter Five

Implementation and Evaluation of MAC and MCAC Algorithms

5.1 Introduction

In this chapter, we show the implementation of the proposed algorithms (MAC, MCAC), and explain their used thresholds. We start by describing the main functionalities of the general Graphical User Interface (GUI) main form which the user interacts with when running the developed algorithms. Mainly, we describe the functionalities of the form components like text boxes, menus and buttons. Moreover, the performance evaluation measures such as cross validation, one error-rate, etc, used in the experiments are described in Section 5.3.

The experimentations of our algorithms (MAC and MCAC) and the other popular AC and rule based classification techniques are conducted after setting up the experiments thresholds (*minsupp*, *minconf*). The data sets utilised in the experiments are related to UCI (20 data sets) and real scheduling problem called the trainer timetabling (8 data sets). We deeply investigate the classifiers generated by our algorithms and the other considered ones against the data sets using known evaluation measures such as predictive accuracy, classifier size, Label-Weight, Any-Label, etc. The large numbers of experimentations mainly focus on the predictive and knowledge quality. Further, we show the pros and cons of our algorithms. Lastly, we summarise the chapter in Section 5.7.

5.2 Implementation of MAC and MCAC

The proposed algorithms have been implemented in Java. The main base class is named “Mac” and it is included in a package called “data mining”. This package contains other class implementations for the proposed algorithms lifecycle starting from data initiation

and ending with test data evaluation. Examples of other main classes which we use are "Rules" for rule generation, and "Column" for discovering frequent ruleitems. For each class, there are a number of main methods implemented to perform the different tasks in it. For instance, "printRankedRules" is a method impeded in the "Rules" class. This method computes the confidence value for a ruleitem. More information on the class implementation source code can be found in Appendix B.

The simple interface of MAC and MCAC is shown in Figure 5.1 in which there is a main menu which the user can use to perform its designated task. The menu headers are "File", "DataMine","Rules"," Cross Validation (CV)", and "Help". Each menu has a number of choices which the end-user can select from, and when a menu choice is selected, this triggers the right method and interface. For example, when the "CV" is chosen by the end-user, this enables him to specify the number of folds and the number of repetition times in a pop up sub-form (box). More details on the interfaces built in this implementation are given in Appendix A. The MAC algorithm implementation shown in Figure 5.1 was the starting point for the MCAC algorithm with additional classes and methods responsible for producing the multiple label rules and the group based prediction. Primarily, the end-user can use the "Iteration" input box in the top of Figure 5.1 to identify the number of classes he wishes to generate using MCAC algorithm.

The proposed algorithms have a user friendly interface as indicated before which enables the end-user to input tuneable thresholds. Hereunder is a quick look of interface thresholds/ parameters for MAC and MCAC algorithms:

The upper interface thresholds/parameters

- Support "text box": The end-user can input the minimum support threshold. This threshold is used to control items that can potentially participate in rules generation.
- Confidence "text box": The end-user can input the minimum confidence value. This value can determine rules that are candidate for the classifier building step.
- Generate "Button": This button is pressed once the training data set is uploaded and the thresholds values are input by the end-user. Once pressed, the classifier model is generated and saved in the program in which end-users can retrieve any time they wish.

- Iteration “text box”: This is an optional parameter that has a numeric value. It is used only for MCAC when the user wishes to produce a multiple label rules of any length. So if the end-user sets this parameter to 3, then he wishes to produce rules that contain three class labels in the rule’s consequent, if any.

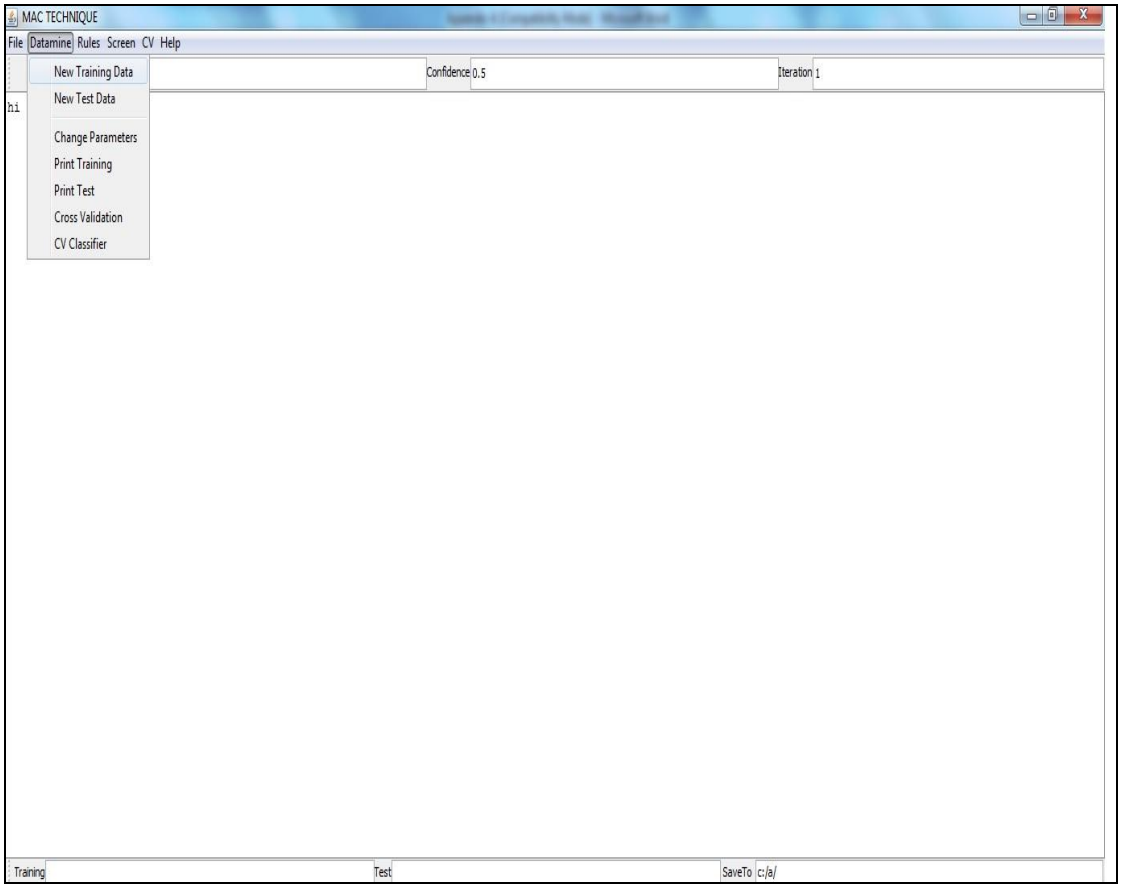

Fig. 5.1 General working environment of MAC and MCAC algorithms

The lower interface thresholds/parameters

- Training “text box”: The designated physical location of the training data file once the end-user has selected it.
- Test “text box”: If the end-user wishes not to use the cross validation as a testing method, then in this text box the location of test data set selected by the end-user will appear.
- Save-To “text box”: In this text box, the end-user can specify the designated location of the classifier model after it’s generated. Please note that this text box is activated only when the test data is used.

A comprehensive example that takes us through the steps in this implementation is provided in Appendix A.

5.3 Measures used for Evaluating MAC and MCAC

In this section, we discuss the different measures that we have utilised to evaluate the performance of both MAC and MCAC algorithms. In particular, we start with a common testing method in data mining called cross validation that we adapt during the classification step to derive the different measure results. We then explain common measures used in data mining literature for single label classifiers performance evaluation such as predictive accuracy, error rate and the number of rules produced. Further, we utilise measures based on predicting class probabilities rather than the actual class itself for MCAC algorithm which have been used in AC literatures like Label-Weight and Any-Label. Overall, this section sheds the light on the evaluation measures that we use to figure out MAC and MCAC performance.

5.3.1 Cross Validation

The majority of classification algorithms when applied against data sets measure the derived classifier's performance with one-error rate (Witten and Frank, 2002). In using this measure, the classifier forecasts the class of each test data and compares it with the actual class. When there is a match between the predicted class and the actual class of the test data this counts as a correct classification (hit). Otherwise it is counted as a misclassification (miss) or an error. The one-error rate is just the numbers of misclassifications made over the test data divided by size of the test data, and it measures the predictive performance of the classifier.

The common testing method of deriving the one-error rate of a classification algorithm in data mining given a single and fixed data is tenfold cross validation. Ten-fold cross-validation is utilised to evaluate the classification models and to produce an error rate of an experiment. It initially divides the input data arbitrary into ten parts in which nine parts are used to learn the rules and the remaining hold out part is utilised to test the rules predictive quality. The procedure is repeatedly invoked ten times on the input data and the derived results (one-error rates) of all runs are then averaged.

Random shuffling is performed in a way to guarantee class representation in each part of the training sets and the single testing set. This is called stratification which guarantees

that all classes are available when the split is executed. Sometimes a single tenfold cross validation might not be enough to derive the final results and thus we have provided in the implementation of MAC and MCAC an option that enables the end-user of specifying the number of tenfold cross validation that he wishes to execute on a data set.

5.3.2 Single Class Evaluation Measures

There are many common single class evaluation methods in classification among the most important ones is one-error rate (Witten and Frank, 2002). For single class problems, we focus in this thesis on evaluation methods related to AC approach mainly one-error rate and accuracy.

5.3.2.1 One-Error Rate

As briefly stated earlier, one-error rate evaluation measure is mainly employed to test the predictive performance of the AC classifiers (Hammoud, 2010). In utilising this method, the classifier derived forecasts the class of a test data, if it is identical to the actual class assigned by the expert, it is counted as a correct classification. If no match it is counted as a misclassification. The proportion of errors from the size of the data set gives the inclusive error rate on this data set. So when the classifier error rate is high this can be considered as a low quality result, otherwise it will be considered a good predictive quality result. The one-error rate is given in Equation (5.1).

5.3.2.2 Classification Accuracy

The classification accuracy is opposite to one-error rate evaluation measure in which it represents the proportion of cases that have correct classification from the size of the data set. A case in the test data is considered correct classification if its actual class matches the predicted class assigned by the rule. A high classification accuracy rate indicates that the classifier is accurate in prediction. The accuracy rate is given in Equation (5.2) based on the data results of the classification process.

$$One\_error(\%) = 1 - Accuracy \tag{5.1}$$

$$Accuracy(\%) = \frac{|TP + TN|}{TP + TN + |FP + FN|} \tag{5.2}$$

Where true positive (TP) denotes cases that have been assigned "Yes" class and their actual class is "Yes", whereas a false positive (FP) denotes cases that are wrongly given class "Yes" (positive class) when they are actually "No" (negative). A false negative (FN) happens when cases are incorrectly labelled as "No" (negative) when it is actually should be positive class. Finally, true negative (TN) occurs when cases are correctly assigned "No" class and they are actually having negative class.

5.3.3 Multiple Class Evaluation Measures

Using single class evaluation measures like accuracy and one-error rate for application data that generate classifiers having rules with multiple classes is only appropriate for the top ranked class in the rules. So to assess the predictive power of a classifier having rules some of which are connected with multiple classes there are other measures in AC mining that were proposed in (Thabtah, 2007). These methods are called Label-Weight and Any-Label which are explained in this section. These evaluation methods do not actually predict the class themselves rather they compute probabilities (weights) for each possible class connected with a rule. During classifying a test case, these methods assign the appropriate probability to the test case based on the class position in the rule. The partial classification of test cases is allowed to avoid the full hit/miss and the probability of the rule's class is given to the test case. Then, this probability is used in the computation of the overall classifier performance.

The mathematical notations used in the definitions of Label-Weight and Any-Label evaluation measures are introduced. Let T be a test data set with m rows t_1, t_2, …, d_m, and C be the class labels set. Each test case t is associated with an actual class $c(t)$. A classifier is a function CL: $T \rightarrow 2^c$, where for $t \in T$, $CL(t) = \langle cl^1(t), cl^2(t), \ldots, cl^{v(t)}(t) \rangle$. The number of times $cl^1(t)$, $cl^2(t)$, …, $cl^{v(t)}(t)$ in the input training data are $f^1(t)$, $f^2(t)$, …, $f^{v(d)}(t)$, respectively.

5.3.3.1 Label Weight

When a rule R is connected with more than one class in its consequent, each class in R may participate in classifying the test case based on its position within R. Thus, Label-Weight can be seen as a fair evaluation measure that avoids the black and white scenario in binary classification. Since the rule's body may be connected with more than one class with different frequencies. Then each class in this rule is given a weight/ probability that denotes

its frequency in the training data with the rule's items (body). Based on this information and the mathematical notations presented in Section 5.3.3, the Label-Weight can be represented as

$$\sum_{t \in T} \sum_{\{i : h^i(t) = c(t)\}} \left(f^i(t) / \sum_{j=1}^{v(t)} f^j(t) \right) \qquad (5.3)$$

The Label-Weight measure considers a test case correctly predicted (hit) if any of the class labels within the rule matches the test case class and it assigns the probability of the fired rule's class to the test case. This probability is used when computing the accuracy for the algorithm's classifier.

5.3.3.2 Any Label

Any-Label evaluation measure is a more optimistic measure than Label-Weight since it considers a test case 100% covered when any of the rule's class matches the actual test case class. That is why it is an optimistic method that often leads to high accuracy on application's data which seeks for a relevant class not necessarily the highest frequency class of the rule. An example of such application is the scheduling data sets generated by the hyperheuristic in which the hyperheuristic is interested in any class (local search method) to apply during constructing the schedule as long as this class has a positive impact on the current schedule. More details on the scheduling problem are given in Section 5.6.1.

In classifying the test case, any of the rule's class can be considered a hit when it matches the test data's actual class regardless of its position in the rule. So once the classification process terminates, the Any-Label method computes the number of hits according to equation 5.4 below

$$|\{t \in T: \exists i \text{ with } h^i(t) = c(t)\}| / m \qquad (5.4)$$

To distinguish between the work of Label-Weight and Any-Label evaluation measures, consider the rule R: $X \wedge Y \rightarrow l_1 \vee l_3$ where attributes value (X, Y) is associated 30 and 20 times with class labels l_1 and l_3 in the training data set respectively. The Label-Weight assigns the predicted class's probability to the test data if the predicted class matches the actual class of the test data. Whereas, Any-Label method assigns "1" to the test data in the same scenario when any of the rule's class matches that of the test data. So for *R* if the test

data actual class is l_3 the Any-Label method assigns the test data "1". Whereas, Label-Weight assigns R's class probability (20/50) = 0.40 to the test data in the same scenario.

5.3.3.3 First / Top Label

Sometimes researchers would like to assess the upside and downside of the first class in the multi-label rule (top-class). This is since there are many applications that only require the best class allocation which normally placed as the first class in the multi-label rule. So, First-Class evaluation measure takes into consideration only the class located at the first position in the rule and allocates it's weight to the test data during the classification step. This method disables all other class labels in the multi-label rules from taking any part in the classification process of test data. Normally, the multi-label rules algorithm is treated as a single label one when using this method. In classifying the test data, when there is a match between the rule class and the test data class the algorithm assigns the test data 1 or the rule's class probability. When classification process terminates, the First-Class measure computes the number of correct classification as below:

$$|\{d \in T: h^1(t) = c(t)\}| / m \qquad (5.5)$$

5.4 Experimental Settings

In all experiments, tenfold cross validation testing method has been employed for fair evaluation of the classifiers derived by the algorithms considered and to reduce overfitting. The experiments were conducted on an I3 machine with 2.3 Ghz. Five dissimilar classification algorithms which utilise a variety of rule learning methodologies have been considered for contrasting purposes with MAC. These algorithms are CBA (Liu, et al., 1998), PART (Frank and Witten, 1998), MCAR (Thabtah, et al., 2005), RIPPER (Cohen, 1995) and C4.5 (Quinlan, 1993). For MCAC, two additional AC algorithms named MMAC and Rank-Label have been chosen.

Our selection of the above classification algorithms is because firstly all these algorithms are rule based ones, e.g. they generate rules in the form of "If-Then" for fair comparison. Secondly, the chosen algorithms use different learning strategies in discovering the rules. The learning strategy exploited by CBA is based on Apriori candidate generation function where frequent ruleitems are produced in level wise search based on the *minsupp* threshold. On the other hand, MCAR uses vertical mining to compute the

ruleitems's support and confidence which in turn are used to decide whether the ruleitem is in fact a rule.

RIPPER is a rule induction algorithm which applies exhaustive search to find the rules (Cohen, 1995) and C4.5 uses information theory measure named Entropy to construct decision tree classifiers. The choice of which attribute should go into the root node in the tree is performed by computing the IG for each available attribute in the training data set. Then C4.5 chooses the attribute with the largest gain in an iterative process. On the other hand, PART algorithms uses a hybrid approach based on decision trees and rule induction learning strategies to construct the classifier. Finally, MMAC discovers the rules from different part of the training data to build the classifier and Label-Rank is an enhancement on MMAC that allows a training data to be used once in generating a rule.

We have set the *minsupp* and *minconf* thresholds for the AC algorithms (CBA, MCAR, MMAC, Rank-Label, MAC, MCAC) to 2% and 50% respectively for all experiments. These are the common thresholds in AC community and have been used successfully for the majority of the algorithms developed (CBA, CBA (2), CMAR, MCAR, CACA, etc). The experiments of C4.5, PART and RIPPER were carried out in Weka software (Weka, 2011), and CBA, Rank-Label and MMAC obtained from their prospective authors. Finally, our algorithms and MCAR were implemented in Java.

Next two sections provide MAC and MCAC experiments and their analysis aiming to assess the performance of the proposed algorithms with respect to the evaluation measures discussed earlier in this chapter.

5.5 MAC Results

We have evaluated MAC against different data sets from the UCI data repository. The selection of the data set was based on different characteristics like data size, the number of attributes, the number of class labels, and the attribute types. We have chosen small, medium and large data sets with various numbers of attributes for fair selection. Table 5.1 gives details about the data sets features.

The focus on the MAC algorithm's results is four different criteria:

1) Classifiers one-error rate (%) .
2) Number of intersections in the frequent ruleitems discovery step (Training phase).

3) Classifier size (# of rules) in normal and sever situations (when *minsupp* and *minconf* are set to very low values).

4) Rule ranking formulas and their impact on classification accuracy and the number of rules in the classifiers.

5.5.1 Error Rate and Intersections Results Analysis

The average error rate obtained by the contrasted classification algorithms on the data sets considered is depicted in Figure 5.2. This figure clearly shows that on average the classifiers generated by MAC algorithm have the least error rate followed by MCAR and CBA algorithms. RIPPER classifiers have derived the worst average error against the data sets followed by C4.5 algorithm. The reason that RIPPER generated the maximum error rate is due to the fact that it employs extensive search in the training phase when it searches for the rule and post rule pruning in which it usually allows limited number of rules to survive. This explains also the small size classifiers generated by this algorithm.

For Figure 5.2, MAC has achieved on average +3.11% and +3.12%, +0.77%, +1.11%, and +1.86% higher predictive accuracy than decision tree (C4.5), rule induction (RIPPER), MCAR, CBA and PART respectively. These results have a good indication that AC approaches usually produce better classifiers than rule induction, PART and decision tree approaches in terms of error rate.

A possible reason for the increase in the accuracy for the MAC algorithm over MCAR

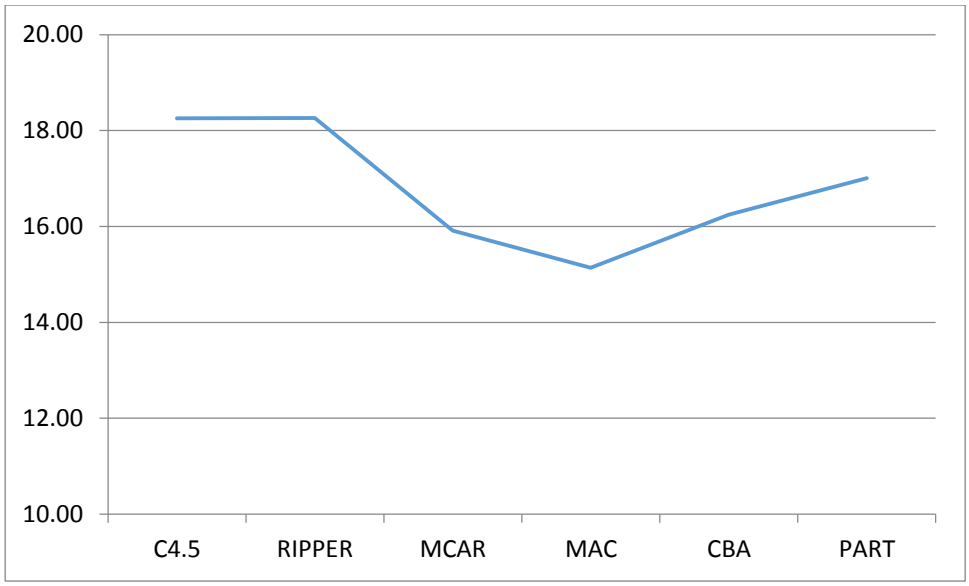

Fig. 5.2 Average one-error rate (%) for the contrasted algorithms derived from the UCI data sets

and CBA is mainly due to that MAC utilises a class assignment procedure that assigns the test data the suitable class based on more than one rules rather than just a single rule as in CBA and MCAR. This means the class of the group having the largest number of rules applicable to the test data is assigned to that test data, which makes the prediction decision more appropriate and thus may slightly enhance the accuracy rate of the classifier.

To elaborate further on the error rates produced by all algorithms, Table 5.1 contains the data sets we consider and the error rate figures of the contrasted algorithms. This table also shows the characteristics of each data set utilised in the experiments including the data set name, the data set size (number of training cases) and the number of class labels per data set. It should be noted that there are small (Labor), medium (Tic-tac) and large (Mushroom) data sets so that we can derive general conclusions on the performance of MAC. After analysing Table 5.1 we found out that MAC algorithm scales well if compared to common AC and other rule based classification data mining algorithms. Specifically, the won-lost-tie

Table 5.1 Error rate (%) of MAC and other AC and rule based algorithms

Data set	Size	# of Classes	C4.5	RIPPER	MCAR	MAC	CBA	PART
Austrad	690	2	13.91	14.34	14.04	12.28	13.22	15.94
Autos	205	7	33.66	43.94	36.10	36.54	39.46	38.54
Balance-scale	625	3	35.68	25.44	14.30	14.72	18.68	18.68
Breast	699	2	5.44	4.58	5.36	5.50	6.76	6.16
Cleve	303	2	23.77	22.45	18.54	20.63	16.90	18.82
CRX	690	2	14.64	15.08	14.04	13.18	14.79	15.08
Diabetes	768	2	26.18	23.96	22.31	22.80	24.13	24.74
Glass	214	7	33.18	31.31	24.76	24.80	23.47	31.78
Heart-s	294	2	18.71	21.77	18.80	19.74	18.13	21.43
Hybothroid	3772	4	4.66	4.66	6.30	6.32	7.68	1.49
Irisd	150	3	4.00	5.34	7.06	5.74	6.69	6.00
Kr-vs-kp	3196	3	29.76	29.76	24.88	22.78	27.64	28.07
Labor	57	2	26.32	22.81	16.49	14.04	13.67	21.06
Led7	3200	10	26.44	30.47	28.10	26.78	30.53	26.44
Lymph	148	4	18.92	22.98	26.08	23.00	25.57	23.65
Mushroom	8124	2	0.23	0.10	0.26	0.08	1.88	0.20
Pima	768	2	27.22	26.70	24.44	25.72	25.42	24.74
Tic-tac	958	2	16.29	3.03	1.02	0.22	1.04	7.42
Vote	435	2	11.73	12.65	13.60	13.80	14.34	12.19
Wine	178	3	5.62	7.31	4.26	4.07	5.04	6.75
Zoo	101	7	6.94	14.86	13.46	5.22	6.14	7.93

record of MAC against C4.5, RIPPER, PART, CBA, and MCAR are 12-8-0, 12-8-0, 14-6-0, 14-6-0, and 10-10-0 respectively.

5.5.2 Classifiers Size and Rules Results Analysis

A deeper examination on the numbers of rules produced by all algorithms and in particular MAC and the other AC algorithms (CBA, MCAR) were performed against the UCI data sets. We have considered two scenarios, one using standard support and confidence (*minsupp* 2%, *minconf* 50%), and one with lower support and confidence (*minsupp* 1%, *minconf* 10%) since we would like to identify the behaviour of the MAC algorithms in normal and severe cases.

The average number of rules produced against the data sets by the AC algorithms for normal scenarios are shown in Figure 5.3. It is clear that MAC algorithm often generates less number of rules than MCAR. Specifically, and in normal circumstances, MAC produced on average 52 rules on the data sets considered resulting in 10.6 less number of rules than MCAR algorithm. CBA derived the least number of rules among the AC algorithms and on average it generated 6 and 16.6 less number of rules than MAC and MCAR respectively.

The primary reason that CBA normally produces the least size classifiers for the AC family of algorithms is because this algorithm uses two types of rule pruning. The first one is an early pruning in which pessimistic error measure from information theory is used to cut down any negative correlation rules. Then, once rules are generated, and during constructing the classifier, CBA removes any rule that does not cover correctly the training case using database coverage method. On the other hand, MCAR and MAC use only one rule pruning technique during constructing the classifier that discards the rule when it has no training data coverage. Though, MAC usually generates smaller size classifiers than MCAR since it utilises the rule pruning without the requirement of class matching during rules evaluation. To avoid overfitting, this pruning method enables the rule to cover more training data and therefore lesser rules are derived.

Furthermore, in severe cases we further evaluated MAC and MCAR in order to evaluate the difference between "Class matching similarity" condition during rules evaluation and the proposed rule pruning method of MAC. We have set the support and confidence to very low values (1%, 10%) and generated the classifiers from the UCI data sets. The number of rules of MCAR becomes even larger and precisely it derived on average 12.69 more rules than MAC. This indicates that in both normal and severe situations MAC normally extracts smaller classifiers than MCAR. In other words, holding a large number of rules in some cases particularly to predict a limited number of test cases is impractical. There should be a trade-off between the classification accuracy and the size of the resulting classifiers especially when slightly more error can be tolerated in exchange for a more concise set of rules. Overall, MAC achieved slightly lower error rate on the UCI data sets than MCAR and it derived less number of rules leading to smaller classifiers.

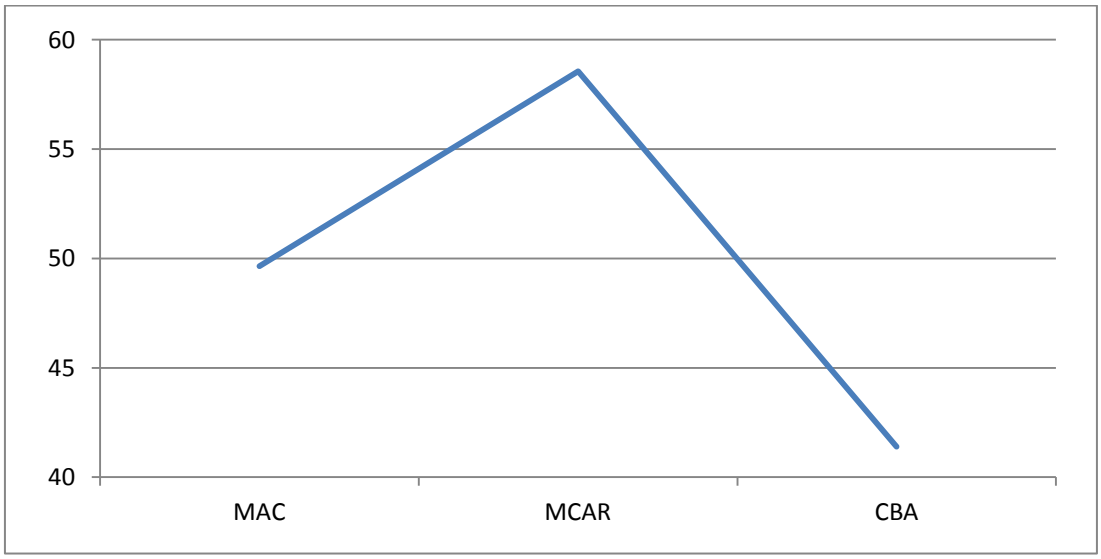

Fig. 5.3 Average number of rules derived by the AC algorithms against the UCI data sets (normal scenario)

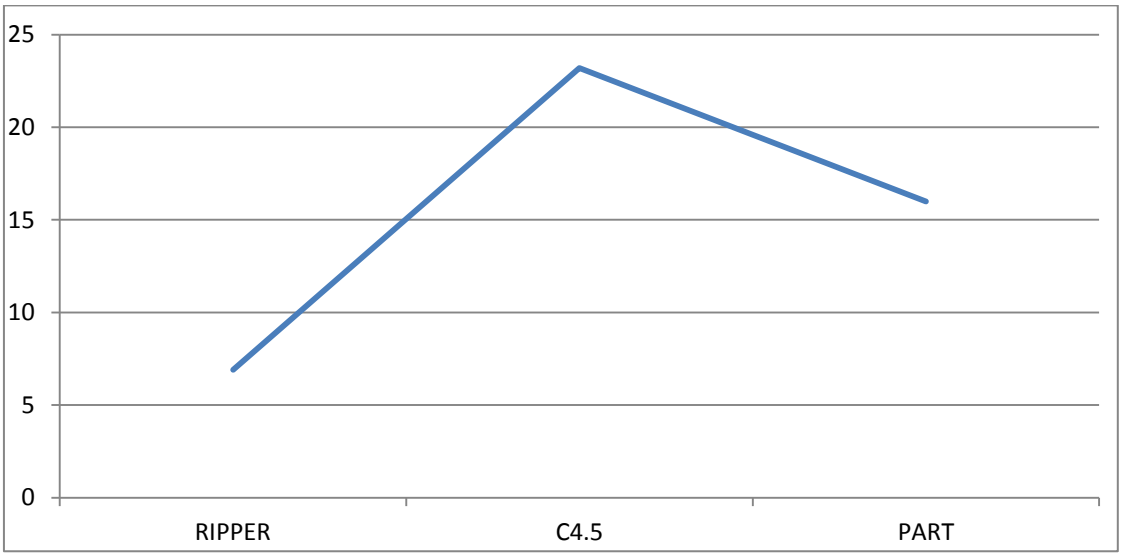

Fig. 5.4 Average number of rules derived by RIPPER, PART and C4.5 algorithms against the UCI data

Figure 5.4 shows the average number of rules derived by the traditional rule-based classification algorithms where it is obvious from the figure that rule induction (RIPPER) generates the least number of rules followed by PART and then C4.5 decision tree algorithm. These classic algorithms employ extensive rule learning and pruning during the training and building the classifier phases. For instance, RIPPER learns the rules greedily and prunes rules using incremental reduced error pruning method (See (Quinlan, 1993) for further details). In fact, (PART and C4.5) employ information theory based pruning based on pessimistic error. This explains the smaller size classifiers if compared with AC algorithms. However, since AC algorithms find additional knowledge concealed within data by testing all possible combinations between the class attribute and other attributes it normally extracts additional knowledge which explains the larger classifiers for MCAR, CBA and MAC over those extracted by classic rule based algorithms.

Overall, we have listed the classifier size generated by each considered AC algorithm and per data set in Table 5.2. The table displays that for most of the data set considered MCAR algorithm derives the largest size classifiers followed by our algorithm and then CBA. These results surely shows how rule pruning is important in AC algorithms that normally suffer from a large number of candidate rules yet generate high predictive classifiers when compared with rule induction and decision trees. The figures in Table 5.2 indicate that the proposed pruning method within MAC has a positive impact on the classifier size by cutting down unnecessary and redundant rules if compared with the

MCAR algorithm. It should be noted that CBA adopts two level of pruning unlike MAC and MCAR as described earlier and employs a candidate generation for rule discovery that reduces the search space. All this explains the less number of rules for the majority of the data sets in Table 5.2 than MAC and MCAR but CBA on the other side generates less accurate classifiers.

The frequent ruleitem discovery step in the training phase has been investigated to expose the saving in the number of ruleitems TIDs intersections during a given iteration of MAC over MCAR. This is since both algorithms are vertical ones for fair comparison. Table 5.3 lists the number of intersections per iteration for a sample of eight data sets. We have selected small, medium and large data sets to derive the number of possible TIDs intersections performed by MAC and MCAR algorithms. The table figures show that MAC saves many intersections during the process of frequent and candidate ruleitems discovery and at any given iteration. Please note that at iteration "1" both MCAR and MAC produce

Table 5.2 Classifiers size of AC algorithms and MAC derived from the UCI data sets

Data set	MAC	MCAR	CBA
Austrad	103	148	142
Autos	55	61	49
Balance-scale	80	75	72
Breast	76	70	52
Cleve	98	101	74
CRX	109	140	147
Diabetes	69	95	40
Glass	19	19	29
Heart-s	26	33	43
Hybothroid	19	18	6
Iris	10	16	5
kr-vs-kp	43	56	40
Labor	11	11	17
Led7	82	159	42
Lymph	54	52	48
Mushroom	47	44	37
Pima	68	94	40
Tic-tac	28	28	28
Vote	74	79	41
Wine	10	11	11
Zoo	15	10	7

the same numbers of frequent 1-ruleitems and no TIDs intersections are required in this iteration so it has been omitted from Table 5.3.

MAC frequent ruleitems technique substantially reduces the number of TIDs intersection for the N ruleitems to produce N+1-ruleitems in an iteration. For instance and for the "Glass" data set shown in Table 5.3, MAC and MCAR possible frequent ruleitems TIDs intersections at iterations 2,3,4,5,6 are (66,32,9,1,0) and (525,885,759,325,42) respectively leading to unnecessary 89% possible intersections for the MCAR algorithm. The total saving in frequent ruleitems TIDs intersections for MAC over MCAR for the sample data sets are depicted in Figure 5.5. These numbers have been derived using the following formula:

$\frac{(X_{MAC} - X_{MCAR})}{X_{MCAR}}$, where X corresponds to the number of possible ruleitems TIDs intersection for all iterations against a data set for the given algorithm.

Table 5.3 The number of ruleitems TIDs intersections per iteration for MCAR and MAC on sample of UCI data sets

			MAC Results			
Data set	**Iteration**	**2**	**3**	**4**	**5**	**6**
Iris		12	3	0	0	0
Balloon		12	4	0	0	0
Lympth		631	909	642	212	0
Glass		66	32	9	1	0
Vote		423	775	897	791	0
Weather		20	3	0	0	0
			MCAR Results			
Data set	**Iteration**	**2**	**3**	**4**	**5**	**6**
Iris		52	19	0	0	0
Balloon		32	16	0	0	0
Lympth		2585	8460	16103	18549	7381
Glass		525	885	759	325	42
Vote		2605	8832	17396	21159	8386
Weather		44	14			

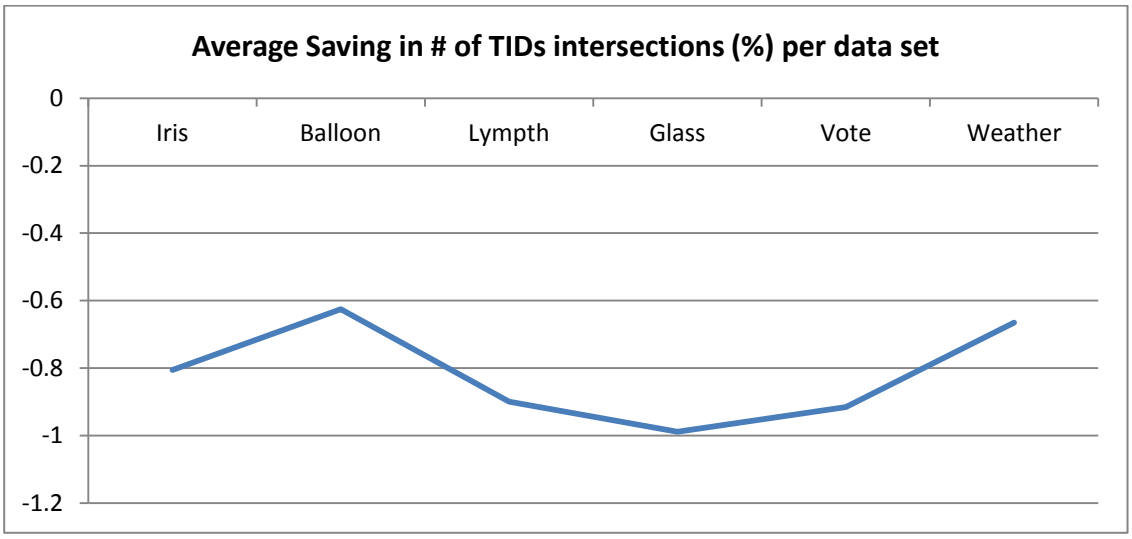

Fig. 5.5 Savings in (%) for MAC algorithm over MCAR in # of TIDs intersections for all iterations per data set

Some numbers in Figure 5.5 are negative because MAC performs less TIDs intersections for the given data set. Overall, and on average MAC was able to cut down the number of TIDs intersections in iteration 2 and later ones by 80% if compared with MCAR. One principle reason for this improvement is due to the fact that MAC algorithm takes only disjoint ruleitems having identical class labels when it performs TIDs intersections unlike MCAR which intersects ruleitems without considering the class and therefore makes too many unnecessary intersections. Lastly, it is worth to note that CBA algorithm that employs the Apriori candidate generation step also merges disjoint N-ruleitems without testing the common class to generate N+1 possible candidate ruleitems, and therefore it may waste so many unnecessary joining.

5.5.3 Rule Sorting Results Analysis

Different criteria in rule ranking have been evaluated on UCI data sets to measure their impact on the classification accuracy and the number of rules generated. Based on the previous research works, e.g. (Liu, et al., 2001; Li, et al., 2001; Thabtah, et al., 2005; Baralis et al., 2008, Chen et al., 2012, Jabbar, et al., 2013), we have considered three main ranking conditions: Rule's confidence, rule's support, and rule's cardinality (shorter length – smaller number of attributes in the rule's body). After warming up experiments that measured the impact of the different ranking conditions orders, we considered only four different combinations of the above criteria as ranking formulas for rules and these are:

- (CONFIDENCE-SUPPORT-CARDINALITY)

- (SUPPORT-CONFIDENCE--CARDINALITY)
- (SUPPORT- CARDINALITY-CONFIDENCE)
- (CONFIDENCE-CARDINALITY-SUPPORT)

The initial warming up test accuracy results on the sample of the UCI data sets revealed that placing rule's cardinality (length) as the first condition when you rank rules usually leads to low classifiers accuracy on test data and thus any formula having rule's cardinality as the first criteria was ignored from further evaluation.

The cardinality criterion is based on the rules that have less number of attributes in its body (general rules). Figure 5.6 shows the accuracy (%) produced by the four rule ranking formulas against the data sets. It is obvious from the graph that (CONFIDENCE-SUPPORT-CARDINALITY) formula outperformed (SUPPORT -CONFIDENCE--CARDINALITY), (SUPPORT - CARDINALITY -CONFIDENCE) and (CONFIDENCE- CARDINALITY SUPPORT) on the majority of the data sets we consider. Particularly, it achieved higher accuracy over (CONFIDENCE- CARDINALITY-SUPPORT), (SUPPORT - CONFIDENCE-CARDINALITY), and (SUPPORT - CARDINALITY -CONFIDENCE) by 5.76%, 12.75% and 13.94% respectively.

These results reveal that rule's confidence is the most fundamental criterion in favouring among rules when it comes to measuring the impact of rule ranking on the classifier's predictive power, then rule support and finally rule cardinality. We further measured the

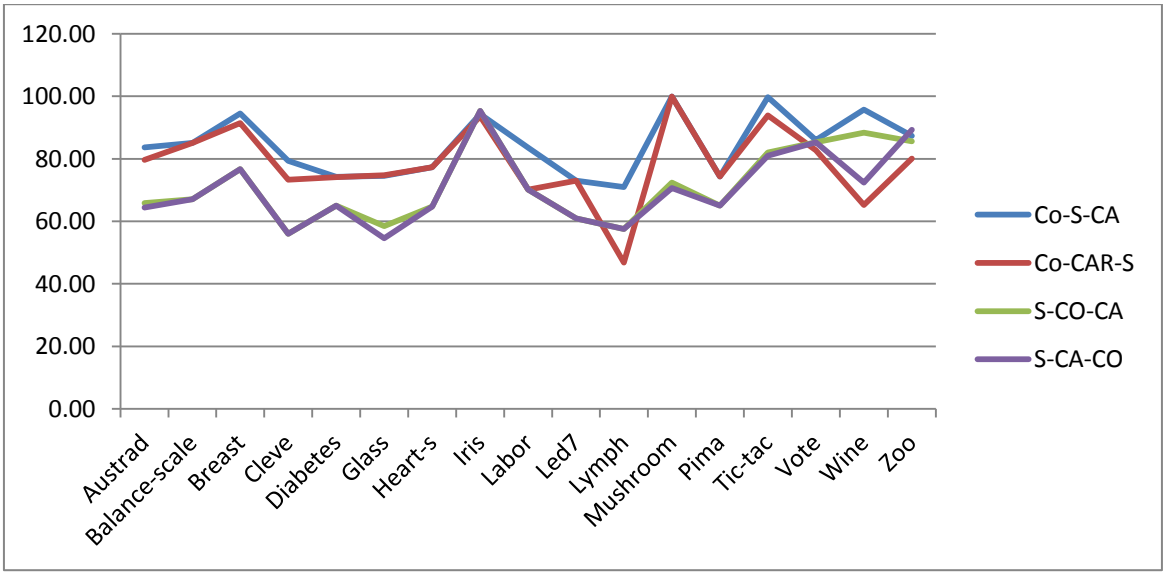

Fig. 5.6 Accuracy (%) derived by the rule sorting formulas

impact of adding a fourth condition which is minority class frequency on top of the best ranking formula which resulted in a new ranking formula. The average accuracy rate on the same data sets after the addition the minority class frequency parameter enhanced by 1.56%.

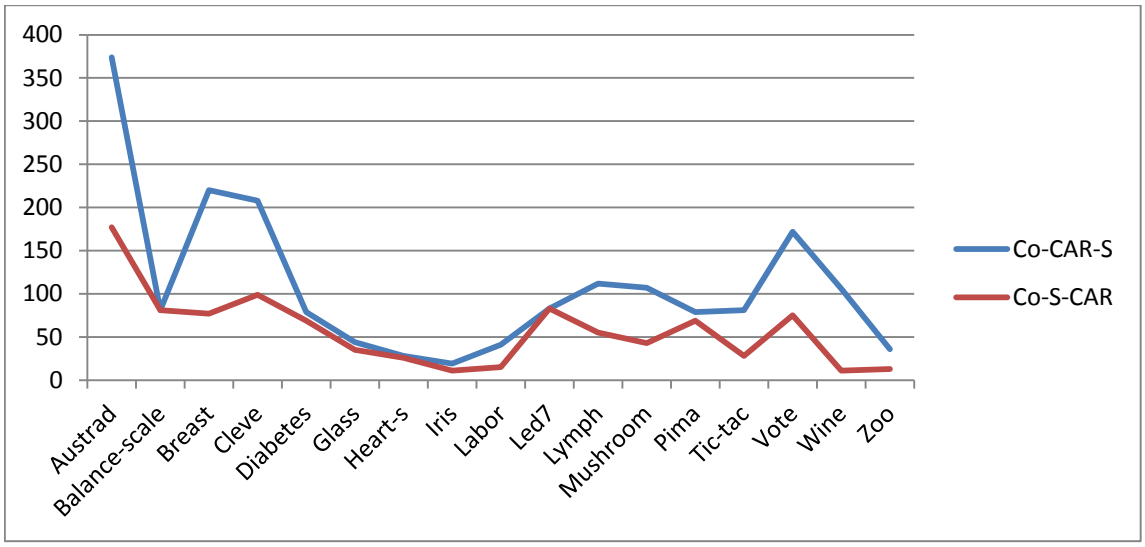

Fig.. 5.7 Number of rules derived by the best two rule sorting formulas

This indeed reveals the crucial influence of adding tie breaking parameters on the process of rule ranking within AC mining.

Moreover, we measured the effect of rule ranking on the number of rules generated in Figure 5.7. Precisely, this figure displays the number of rules generated when using the best two rule ranking formulas in accuracy performance which are (CONFIDENCE-SUPPORT-CARDINALITY) and (CONFIDENCE-CARDINALITY-SUPPORT). It is clear from the graph that (CONFIDENCE-SUPPORT-CARDINALITY) generates the least number of rules if compared to the (CONFIDENCE-CARDINALITY-SUPPORT). In particular, and for the data sets, the average number of rules generated by (CONFIDENCE-SUPPORT-CARDINALITY), (CONFIDENCE-CARDINALITY-SUPPORT) are 56.88, and 110 respectively. This means that the (CONFIDENCE-SUPPORT-CARDINALITY) method not only produces high quality classifiers with reference to accuracy but also smaller size ones if contrasted with (CONFIDENCE-CARDINALITY-SUPPORT). It should be noted that we use the words “formula” and “method” when talking about rule ranking interchangeably and they refer to the same meaning.

To have an insight look on the behaviour of the above rule ranking methods, Table 5.4 shows the number of times each criterion does not break tie between rules for a sample of

data sets. Column “Conf” indicates the number of rules with identical confidence values, column “Conf&Supp” represents the number of rules with the same confidence and support values. Column “Conf&Supp&Card” depicts the number of rules that have similar confidence, support and cardinality. Values shown in Table 5.4 represent the candidate rules tested by the MAC algorithm during the rule ranking and before constructing the classifier. Table 5.4 shows that support and confidence are not effective enough in distinguishing among rules in most data sets we consider. For the “Austrad” data set for instance, there are 2421 rules with the same confidence as some other rule, with 233 rules having identical confidence and support. There are 10 with the same confidence, support and cardinality as some other rule. These results necessitate considering new tie breaking criterion to further favour between rules during the ranking process.

Table 5.4 Number of times each rule ranking criterion does not break tie between rules

Dataset	Conf	Conf&Supp	Conf&Supp&Length
Austrad	2421	233	10
Balance-scale	17	17	2
Breast	210	134	5
Cleved	695	108	7
Diabetesd	351	121	5
Germand	3443	323	8
Glassd	193	55	6
Heart-s	121	71	5
Irisd	32	21	4
Led7	535	131	7
Mushroom	557	226	7
Pimad	351	121	5
Tic-tac	212	67	4
Vote	845	153	8
Wined	466	56	6
Zoo	136	34	6

5.6 MCAC Results

In this section, we conduct experimentations on a number of classification data sets to evaluate the MCAC algorithm performance. Specifically, we want to measure the performance of MCAC in the following scenarios:

1) The multi-label rule’s role in improving the accuracy of the derived classifiers. Here we compare MCAC classifiers with other AC multi-label rules techniques like

MMAC. Since the class labels associated with the rule in MCAC's classifier are of disjunctive form we have selected only algorithms that derive rules in a disjunctive form for fair comparison.

2) The numbers of the multi-label rules and their importance to the decision makers. Here we show the novelty of MCAC in deriving multi-label classifiers from single label data sets.

3) Relative accuracy which shows the difference in % between MCAC in predicting test data if contrasted to other classification algorithms.

4) The top ranked class (First class) in the MCAC's classifier rules and compare its predictive accuracy with common AC and other classification algorithms.

The main evaluation measures used in the experiments are prediction rate (Accuracy) / First-Label, (Label-Weight, Any-Label), and classifier size (number of rules). A number of algorithms have been contrasted with MCAC with respect to the above mentioned evaluation measures. This section is divided into two sub-sections:

- Real world data sets experiments:

  A number of solutions devised by a heuristic method called the hyperheuristic that consists of over 3800 instances have been utilised. The hyperheuristic is a general purpose method that normally used to solve complex scheduling problems by choosing the right local search neighbourhoods at each iteration while building the schedule (Cowling and Chakhlevitch, 2003; Chakhlevitch and Cowling, 2008). The hyperheuristic data have been generated for a timetabling problem named the "trainer scheduling" (Chakhlevitch and Cowling, 2008; Thabtah and Cowling, 2007). In the data sets given, and for each training example, there could be more than one local search methods that could improve the objective function by the hyperheuristic. More details on the data sets of the trainer scheduling problem are given in Section 6.1.

- UCI data sets experiments: We have utilised a number of data sets from UCI data repository (Merz and Murphy, 1996). The main reason for using such data sets is to evaluate the first ranked class associated with the rule in the MCAC classifier for fair comparison.

5.6.1 Trainer Timetabling Data Description

A hyperheuristic can be seen as a general purpose search method that manages the selection of the local search or simpler methods during building a solution / schedule for a scheduling problem. A local search method is a rule that when applied by the hyperheuristic usually gives a change in the solution (positive / negative). The local search methods are normally human ways of making the solution like removing an event or adding an event (Chakhlevitch and Cowling, 2008). The way the hyperheuristic works are depicted in Figure 5.8 where at each step, it applies a local search method that yields the best performance (normally measured by an objective function) on the solution after evaluating all available local search methods.

We consider in this thesis solutions (data) collected from a trainer scheduling problem for a large financial company. This problem involves a number of trainers, events, and locations to be scheduled over a specific period of time. The aim is to build a comprehensive timetable of training courses using a predefined number of trainers over a number of geographical locations in a specific period of time. A more detailed description of the problem is presented in (Cowling and Chakhlevitch, 2007).

The process of detecting the right local search method while the hyperheuristic is constructing the solution is a typical classification problem where a number of different local search methods are tested to enable the hyperheuristic to choose the ones having the positive impact on the current solution. So if we have 50 different local search methods, traditionally the hyperheuristic must test all of them in order to find out the one or the set of methods that have the largest positive impact. In classification data mining, we would like to discover the correlated sequences of local search methods in order for the hyperheuristic to guess directly the correct local search method at any given decision point during making the schedule. The correlations between the local search methods are represented as simple chunk of knowledge in the form of "If-Then" rules and are in fact the classification system.

Now the role of classification here is that an algorithm can be employed to discover the correlations among the applied hyperheuristic in old solutions (data sets) and devise them as rules. The training data or the old solutions are the main source to train for rules and they consist of sequences of applied local search methods where the class is the set

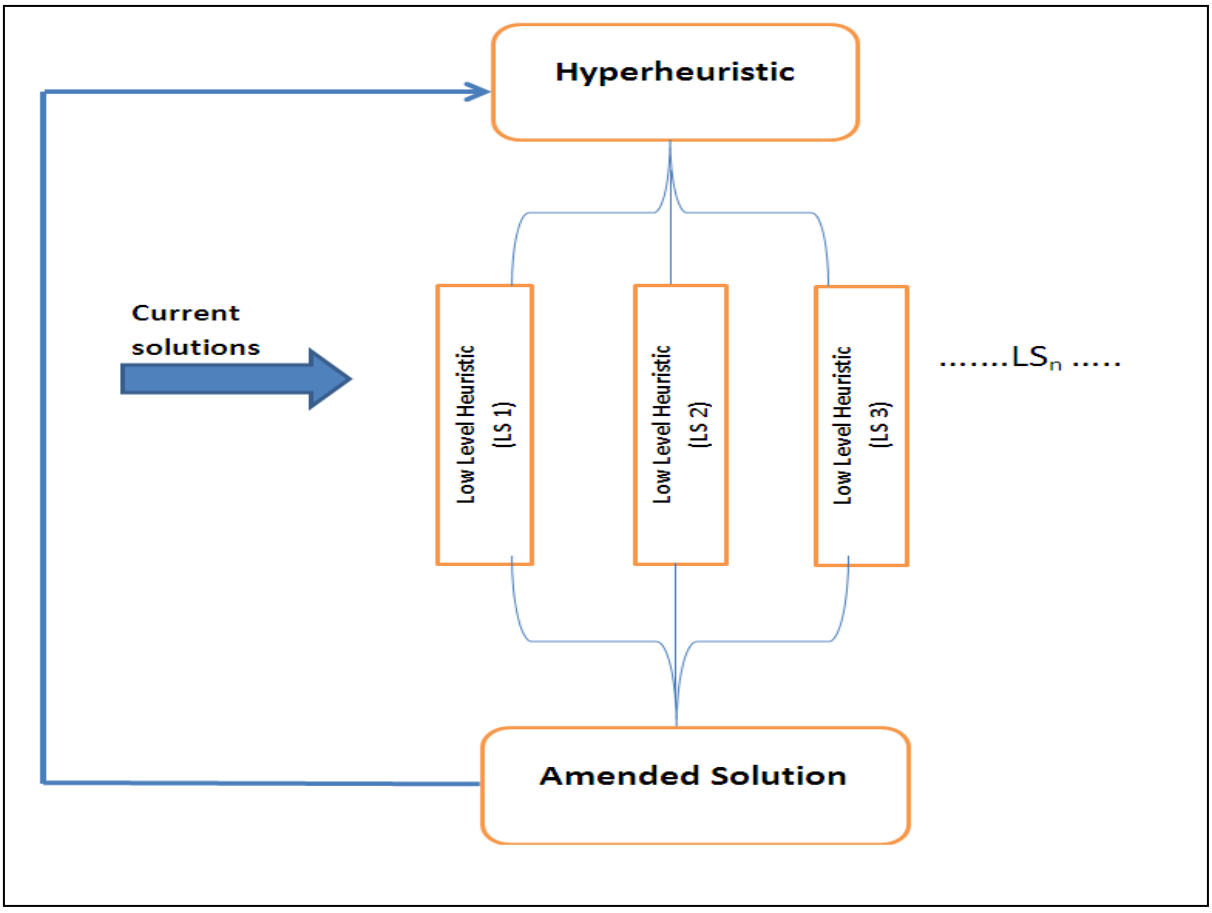

Fig.5.8 The hyperheuristic way of choosing the local search methods in building the trainer timetable/schedule

of local search methods that when applied improve the schedule at a given iteration. A sample of the rules we are looking for look like: $(LS_5 \wedge LS_2 \wedge LS_3 \rightarrow LS_1 \vee LS_{10})$. This rule denotes that if local search methods (LS_5, LS_2, LS_3) are applied in sequence in three iterations then the most likely local search method that will be applied in the next iteration is either $LS_1 or LS_{10}$.

So our aim is to determine the applicability of our MCAC to the problem of figuring out the set of local search methods to use in a given iteration, using information derived earlier about previously applied local search methods. We are looking to answer questions like "can MCAC discover correlations (rules) from past data that may direct the hyperheuristic search in new data by choosing the right local search methods to apply?".

These correlations will be employed to decide the class "local search methods" while building new solutions in the future by the hyperheuristic.

We have obtained a number of solutions (data sets) consisting of over 3800 instances from (Cowling and Chakhlevitch, 2007; Thabtah and Cowling, 2007) to achieve our aim. The data sets contain sequences of ten local search neighbourhood methods that the hyperheuristic has utilised while building seven different solutions in the past. Each data set contains different iterations where each training instance is associated with one class among the ten local search methods (1-10). The ten local search methods used in the experiments are denoted (LS1, LS2, LS3, LS4, LS5, LS6, LS7, LS8, LS9, LS10).

A sample of data in a solution for the trainer problem is depicted in Table 5.5 where the last column corresponds to the local search method that was applied by the hyperheuristic since it improved the objective function of the schedule the most. Columns 1-5 represent the sequence of the last five iterations of applied local search methods. Column 6 (Positive LS) denotes the local search method that have positive effect on the schedule if applied.

Table 5.5 Sample data for the trainer scheduling problem

LS -5	LS -4	LS -3	LS -2	LS -1	Positive LS	Impact on Objective. Function?
3	5	3	6	10	7	yes
3	5	3	6	10	9	yes
7	4	9	1	2	10	yes
7	4	9	1	2	3	yes
7	4	9	1	2	5	yes
9	4	3	5	7	6	yes
10	7	2	3	1	6	yes
10	7	2	3	1	8	yes
10	7	2	3	1	1	yes
10	7	2	3	1	4	yes

5.6.2 Results on the Trainer Timetabling Data

Eight different data sets some of which consist of multiple schedules devised by the hyperheuristic for the trainer scheduling problem have been collected. Each data example that belongs to any data set is simply a sequence of applied local search methods by the hyperheuristic at a given iteration. Further, each data set is having six different features and

a number of data examples that had a positive effect on the objective function of the trainer scheduling problem.

We have chosen two AC multi-label rules algorithms named MMAC and Rank-Label to evaluate the performance of MCAC algorithm. The selection of these algorithms are based on the facts that a) They are AC algorithms and b) They generate the same type of output format as MCAC. The performance of the derived classifiers of all considered algorithms is based on two known evaluation measures in AC mining named Label-Weight and Any-Label that have been discussed in Section 5.3.

The difference in classification accuracy for the classifiers also known as relative prediction accuracy between MCAC and (MMAC, Rank-Label) algorithms are displayed in Figures 5.9 and 5.10 respectively. The relative prediction figures have been computed using the following mathematical formulas: $\frac{(Accuracy_{MCAC} - Accuracy_{MMAC})}{Accuracy_{MMAC}}$ and $\frac{(Accuracy_{MCAC} - Accuracy_{Rank-Label})}{Accuracy_{Rank-Label}}$ for both MMAC and Rank-Label algorithms respectively. The ratios in % shown in Figures 5.9 and 5.10 have been derived using Label-Weight evaluation measure. It is clear from the figure that MCAC outperformed MMAC and Rank-Label for the majority of the scheduling data sets. In particular, the won-lost-tie record of MCAC vs MMAC and Rank-Label algorithms are 7-1-0 and 6-2-0 respectively on the Label-Weight measure.

The way MCAC finds and extracts the rules allow it to have more class labels in the rule consequent which positively impacts figures derived by Label-Weight on the classifier. This is since when there are more than one class in the rule this gives the rule more options to select from when classifying test data. So, rather than using one class which can be either a correct classification or a misclassification during class assignment of the test data, MCAC enables rule of having multiple classes which surely

improves the classifiers performance since it allows for partial classification by assigning the rule's class probability to the test data.

The MMAC and Rank-label algorithms have outperformed MCAC on the first data set of the scheduling problem solutions in Figure 5.9 and 5.10. After investigating this data set it turns out that it contains 112 instances yet it derives large number of rules the majority of which are single label having high rank. So when a test data requires classification mainly the single label rules take on this task for this data set in particular. This makes the role of

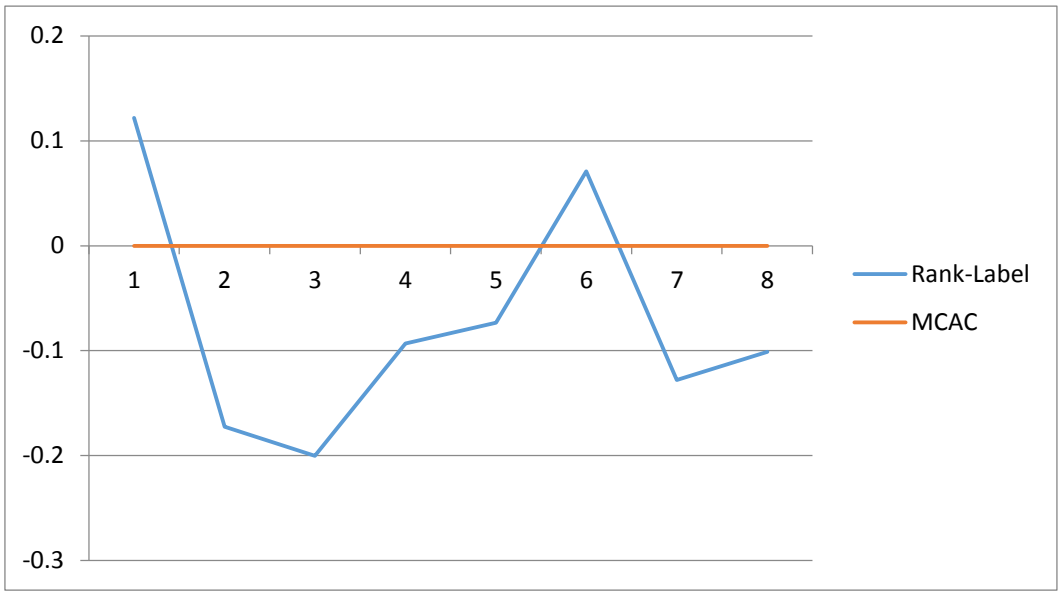

Fig. 5.9 Relative prediction accuracy of MCAC vs Rank-Label on the trainer scheduling data

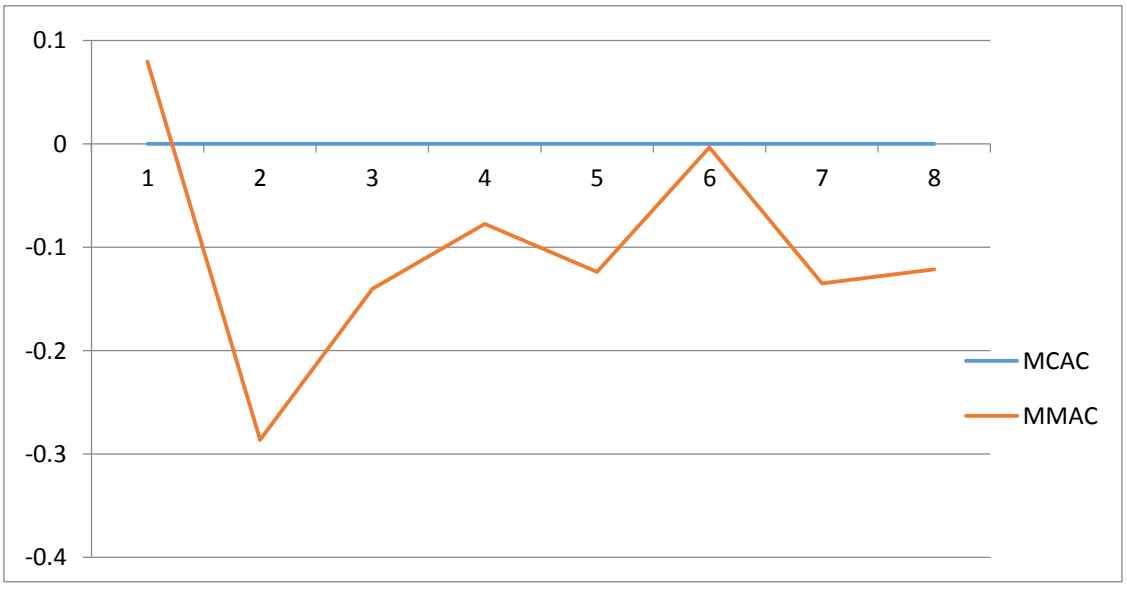

Fig. 5.10 Relative prediction accuracy of MCAC vs MMAC on the trainer scheduling data

the multi-label rules in the classifier for this particular data set limited and therefore little enhancement is gained in accuracy rate. Consequently, when computing the accuracy rate on the holdout block in cross validation, there are mainly hits and misses (0 or 1) since the rules fired are primarily single label ones.

Figure 5.11 shows the classifiers performance of MCAC derived from the trainer scheduling data sets using First-Label, Label-Weight and Any-Label evaluation measures. The figure demonstrates that Any-Label evaluation measures is the best for this type of application since the hyperheuristic does not care about which local search method to choose as long as this local search method had an improvement on the objective function and thus enhanced the current solution or the schedule. The reason for the largest improvement of the classifiers by the Any-Label measure is that this method considers any class assigned to the test data a correction classification and gives "1" to the test data. Therefore, for instance when the multi-label rule having three class labels on its consequent is fired on the test data that has one actual class, any of the rule's class labels can be assigned to that test data. Thus, when the Any-Label accuracy is computed the number of correct classification on test data often larger than First-Label and Label-Weight measures results. All these facts explain the reason for the higher figures devised by the Any-Label

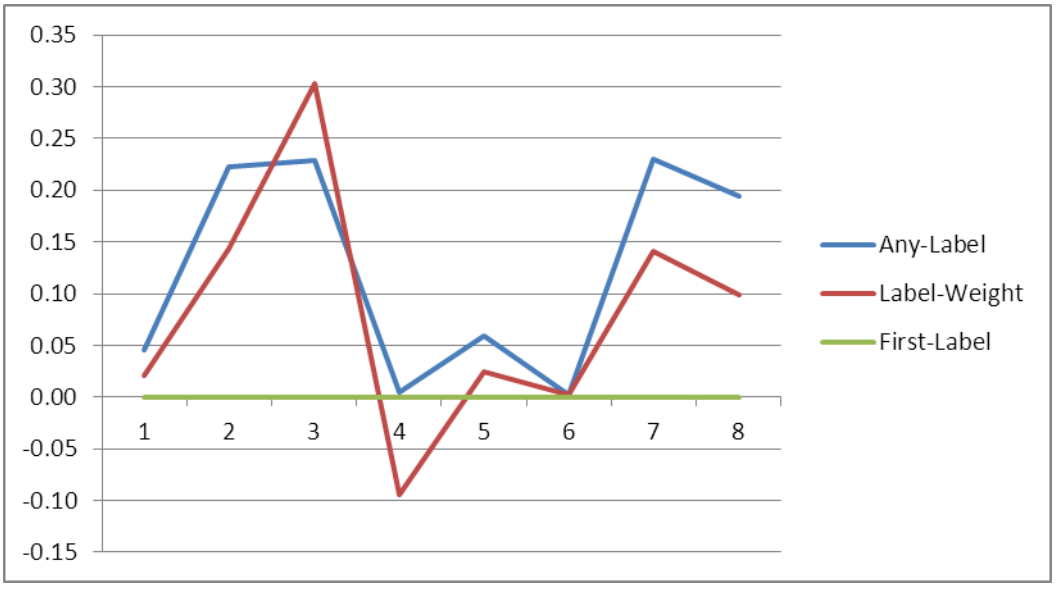

Figure 5.11 The difference in accuracy between (Any-Label, Label-Weight, First-Label) for MCAC algorithm on the trainer scheduling data set

measure on the trainer scheduling application data.

Moreover, the top ranked class for both MCAC and MMAC has been investigated by calculating the First-Label evaluation measure on the classifiers generated by both algorithms. This measure is similar to accuracy in single label classification algorithms. Figure 5.12a demonstrates the numbers of the First-Label for both MCAC and MMAC algorithms and against the trainer scheduling data set. The figure noticeably illustrates that MCAC outperformed MMAC even on the top ranked class in the multi-label rules when it comes to predictive power and for most of the trainer scheduling data sets. This means even the single label version of MCAC is still powerful in classifying test data if contrasted with existing rules AC algorithms.

We have also examined the top ranked class power of MCAC and compared the classifiers derived using only the highest ranked class with other classification algorithms in data mining. Precisely, we have produced the classifiers of four AC and rule based algorithms which are C4.5, PART, CBA, and MCAR and contrasted the difference in accuracy between them and MCAC as depicted in Figure 5.12b. The reasons for conducting such experiments are to further investigate the first class effect on the classifying test data and to validate the new prediction method presented in MCAC. Figure 5.12b indicates clearly that MCAC outperformed C4.5, PART, CBA, and MCAR on the data sets we consider and according to accuracy rate. The won-lost-tie records of MCAC versus C4.5, PART, CBA, and MCAR are 6-2-0, 7-1-0, 7-1-0, and 6-2-0 respectively. A possible reason for the enhancement on the accuracy by MCAC is the classification method employed by our algorithm reduces the use of default class in predicting test cases.

Lastly, we looked at the Any-Label figures produced by both MMAC and MCAC algorithms to seek the one which has the largest gain on the trainer scheduling problem. Figure 5.13 displays the Any-Label classifiers results for the two algorithms in which our algorithm has outperformed MMAC using the Any-Label measure on the derived classifiers from the scheduling data sets we consider. In fact, the Any-Label evaluation measure as stated earlier in Figure 5.11 is most suitable measure for the trainer scheduling application.

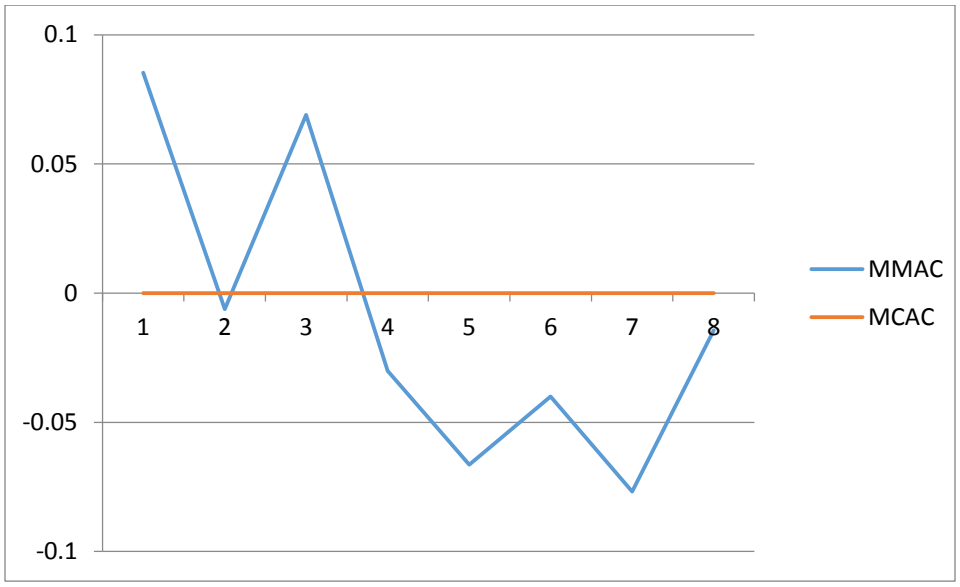

Fig. 5.12a The difference in (%) derived by First-Label measure for the MCAC and MMAC algorithms from the trainer scheduling data

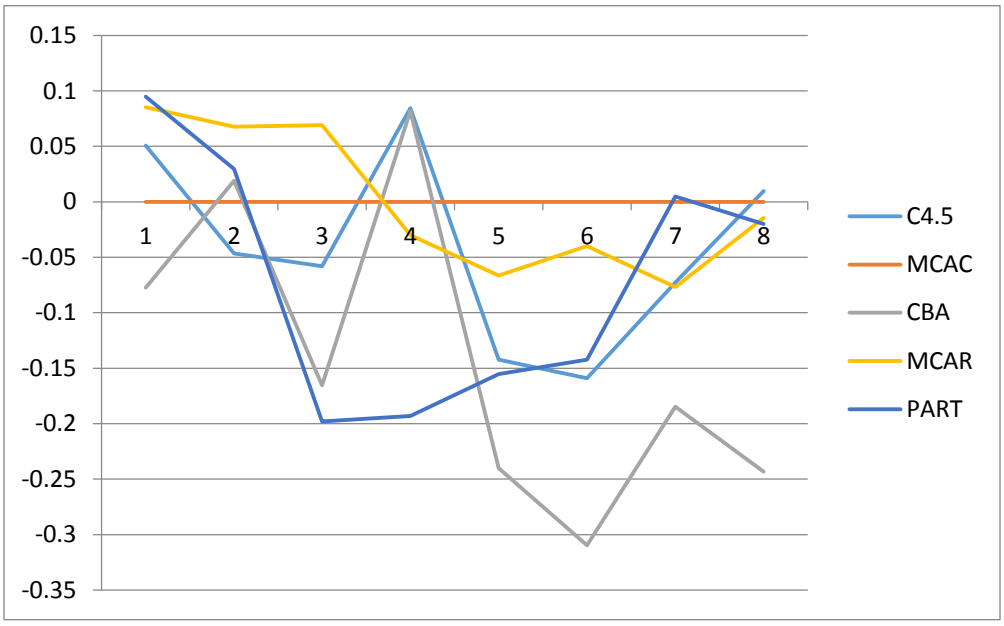

Fig. 5.12b The difference in accuracy (%) between MCAC First-Label and (C4.5, CBA, MCAR, PART) algorithms on the trainer scheduling data

Further, the number of rules have been deeply examined and for each data set for the hyperheuristic. Figure 5.14 demonstrates the number of 1-label, 2-label, 3-label and 4-label rules devised from the data sets we consider by MCAC. It is evident that our algorithm is not only able to derive rules associated with one class but also with a set of class labels concealing important hidden knowledge that other algorithms are unable to find. The fact

that MCAC is discovering rules associated with multiple class labels is a proof on its applicability in real world domain applications such as scheduling. In

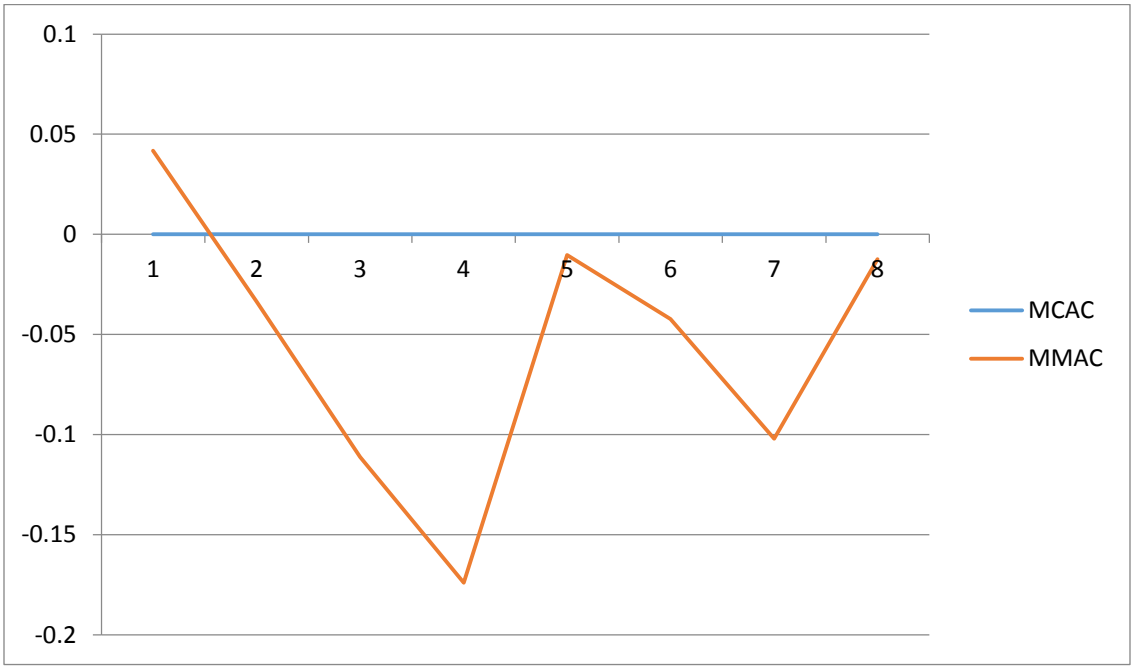

Fig. 5.13 The difference in (%) derived by Any-Label measure for MCAC and MMAC algorithms on the trainer scheduling data

Chapter 6, we show MCAC applicability to another vital application related to web security called "Website Phishing Classification". These new discovered multi-label rules are not only enhanced the predictive power of the classifiers derived as shown in the previous figures but also are crucial for the decision makers. For example, in the trainer scheduling application, the MCAC algorithm is able to offer the hyperheuristic with rules having many options of local search methods to choose from. So unlike the traditional search used by the hyperheuristic in which a local search method is chosen after examining all available local search methods to find out the positive related ones. Now, the hyperheuristic can use the multi-label rules to select one or more local search methods among a smaller set of the discovered rules consequent which significantly reducing the search space of the problem.

Figure 5.14 also shows that most of the rules are connected with one or two labels. So we looked deeply in the data sets to examine the way the hyperheuristic builds the schedule. We found out that the hyperheuristic usually ends up with one or two local search methods at any given iteration that improved the objective function of the schedule.

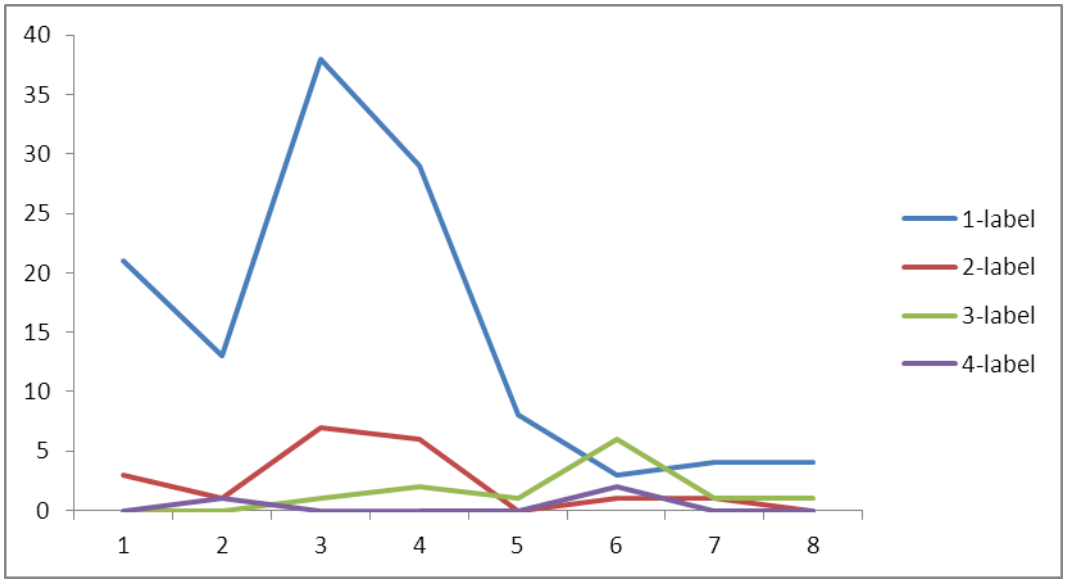

Fig. 5.14 The number of labels associated with rules in the classifiers derived by MCAC from the trainer scheduling data

The number of rules generated by the MCAC and three other classification algorithms are displayed in Figure 5.15 where surprisingly MCAC derived smaller classifiers than decision trees, PART and MCAR on the scheduling data sets. The reasons that MCAC extracted less number of rules are the following:

1) Each feature of the data sets in the trainer scheduling problem has several possible values and therefore decision trees algorithms like C4.5 and PART which also constructs partial decision trees splits many branches at each decision point and during construction of the trees. Many of these branches lead to useless and redundant rules and therefore this explains these algorithms large classifiers when compared to MCAR and MCAC. It should be noted that these results are opposite of the classifier size results presented earlier in this chapter which consequently leads to a conclusion that decision tree based algorithms are not suitable for the problem of selecting the set of local search methods in the trainer scheduling application.
2) The MCAC algorithm was able to combine single label rules into multi-label ones during the rule discovery phase

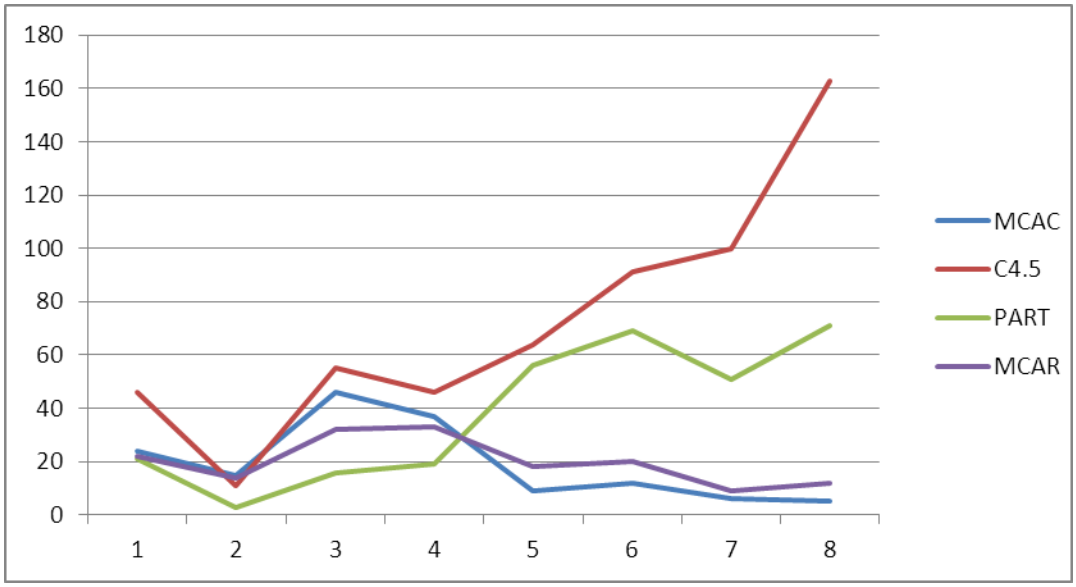

Fig. 5.15 The number of rules derived from the scheduling data set using a number of classification algorithms and MCAC

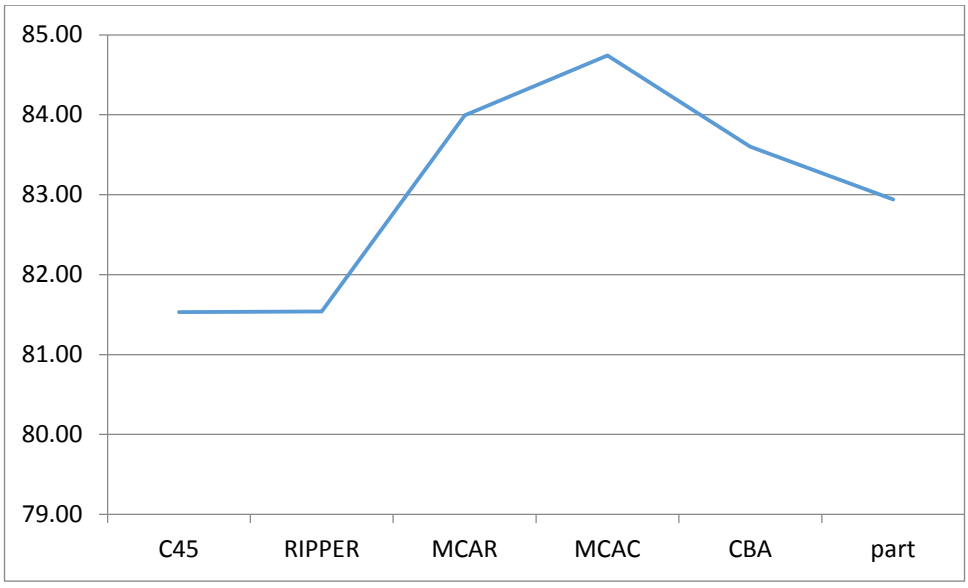

Fig. 5.16 Average classification accuracy (%) for the contrasted algorithms derived from the UCI data sets

5.6.3 UCI Data Results

Different data sets described in Table 5.6 from the UCI data repository have been used to measure the effectiveness of the top label (First-Label) in MCAC. The selection of the data sets was based on different criteria like number of examples contained; attribute types, and the number of classes. We would like to evaluate the predictive strength of the first class

associated with the rule in the MCAC classifier on test examples by comparing its accuracy with those generated by all contrasting algorithms.

Figure 5.16 illustrates the average classification accuracy (%) derived by the considered algorithms on the UCI data sets considered. The figure clearly shows that MCAC algorithm derived on average higher accuracy than the other algorithms. Specifically, MCAC outperformed RIPPER, C4.5, PART, CBA, and MCAR on average 3.18%, 3.02%, 2.09%, 1.00%, and 0.54% respectively with respect to accuracy rate on its top ranked class within the rule. In fact, as we will see soon MCAC algorithm balances between the number of rules in the classifier and the classifier's accuracy for the sake of smaller number of rules yet with high predictive quality.

The increase in the classification accuracy of the MCAC algorithm over the other AC algorithms is mainly due to the prediction procedure that stores rules in two scenarios: a) When the rule body is fully contained in the test case and if this condition is not true b) MCAC uses the first rule (highest ranked rule) that is contained in the test case rather classifying the test case by the default class. This may reduce utilising the default class for predicting test data and decreases the error rate of the classifier.

Table 5.6 displays the data sets description including their name, the number of examples, and the number of classes. In addition, the accuracy of the classifiers for each learning algorithm and for each data set is also depicted in the table. The table shows that MCAC has outperformed the other algorithms in accuracy rate of the first class and for the majority of the data set. Explicitly, the won-lost-tie record of the MCAC algorithm against RIPPER, C4.5, PART, CBA, and MCAR are 13-7-0, 13-7-0, 14-6-0, 13-7-0 and 11-7-2 respectively.

Table 5.6 Classification accuracy (%) of the contrasted algorithms on the UCI data sets

Data set	Size	# of Classes	C4.5	RIPPER	MCAR	MCAC	CBA	PART
Autos	205	7	66.34	56.06	63.90	63.90	60.54	61.46
Balance-scale	625	3	64.32	74.56	85.70	85.28	81.32	81.32
Breast	699	2	94.56	95.42	94.64	95.00	93.24	93.84
Cleve	303	2	76.23	77.55	81.46	79.37	83.10	81.18
CRX	690	2	85.36	84.92	85.96	86.93	85.21	84.92
Diabetes	768	2	73.82	76.04	77.69	74.20	75.87	75.26
Glass	214	7	66.82	68.69	75.24	75.20	76.53	68.22
Heart-s	294	2	81.29	78.23	81.20	81.26	81.87	78.57
Hybothroid	3772	4	95.34	95.34	93.70	94.34	92.32	98.51
Irisd	150	3	96.00	94.66	92.94	94.26	93.31	94.00
Kr-vs-kp	3196	3	70.24	70.24	75.12	77.00	72.36	71.93
Labor	57	2	73.68	77.19	83.51	85.96	86.33	78.94
Led7	3200	10	73.56	69.53	71.90	73.22	69.47	73.56
Lymph	148	4	81.08	77.02	73.92	78.44	74.43	76.35
Mushroom	8124	2	99.77	99.90	99.74	99.92	98.12	99.80
Pima	768	2	72.78	73.30	75.56	74.68	74.58	75.26
Tic-tac	958	2	83.71	96.97	98.98	99.89	98.96	92.58
Vote	435	2	88.27	87.35	86.40	88.00	85.66	87.81
Wine	178	3	94.38	92.69	95.74	94.88	94.96	93.25
Zoo	101	7	93.06	85.14	86.54	93.10	93.86	92.07

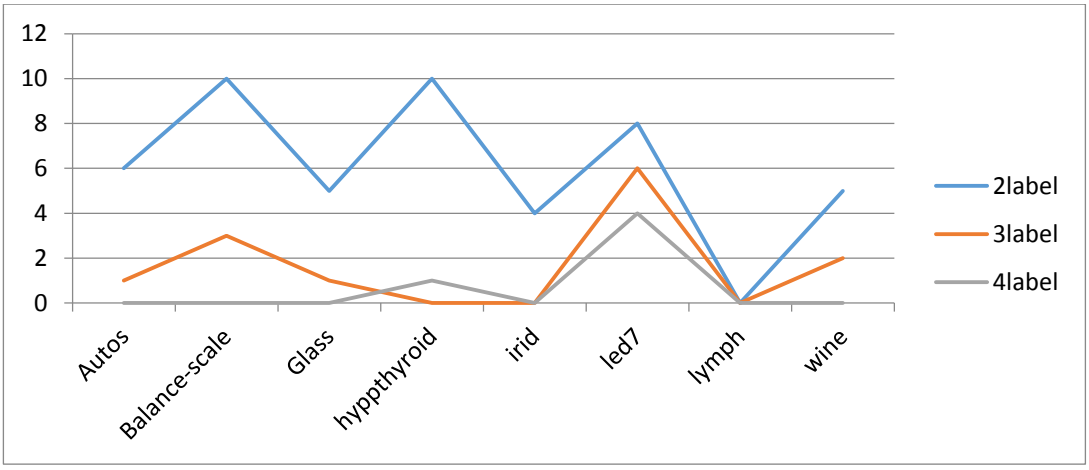

Fig 5.18 the number of multi-label rule derived from some UCI datasets.

For fair comparison, Figure 5.17 demonstrates the number of rules generated by only the AC algorithms on the data sets. It is notable from the figure that MCAR algorithm still produced the largest number of rules followed by MCAC and CBA. The reason for CBA to generate the least number of rules on average for the AC group of algorithms is explained

earlier and briefly because it utilises pessimistic error (Quinlan, 1993) when building the rule and database coverage during building the classifier. Nevertheless, MCAC's classifiers are smaller than MCAR due to avoiding the excessive learning that determines rules goodness aiming to reduce overfitting on the training data set. We consider checking the class of the training data during building the classifier cheating that do not add value to the candidate rules set based on the obtained results. The fact that MCAR requires class similarity is considered by MCAC over learning which may a) increases the number of rules and b) does not guarantee rule goodness since the rule is not yet utilised in classifying test data. MCAC extracted on average less number of rules than MCAR with competitive accuracy rates and little more rules than CBA with an increased accuracy rates.

To signify whether MCAC is able to produce rules with multiple labels from the UCI data sets especially the multi-class ones we ran MCAC against them. Figure 5.18 displays the multi-label rules count generated by MCAC per multi-class data set categorized by the number of labels. Particularly, we show the number of rules that are connected with 2-label, 3-label and 4-label per data set. The results clearly provide evidence that MCAC extracted multi-labels rules that correspond to new knowledge and consequently contribute towards the predictive accuracy of the classifiers.

5.7 Chapter Summary

In this chapter, we highlighted the implementation details of the proposed algorithms in this thesis including the different parameters used and a sample of the set of classes and methods. Moreover, the different evaluation methods to test the performance of MAC and MCAC algorithms such as one-error rate, classification accuracy, Label-Weight and others are discussed.

For MAC algorithm, a number of experiments against data sets from the UCI data repository using different rule based and AC algorithms have been performed. The bases of the experiments are certain evaluation measures like one error rate, classifier size, ranking effect on accuracy, etc. The results obtained reveal that MAC outperformed traditional classification C4.5 and RIPPER algorithms with respect to error rate, and it scales well if compared with known AC algorithms. Moreover, MAC usually generates less number of rules than MCAR in normal (standard *minsupp*) and severe situations (low *minsupp* and *minconf*) on the data sets we consider because of the novel rule pruning method proposed. For some data sets MAC achieved less accuracy than MCAR algorithm though it has

extracted on average 10.6 less number of rules. This indeed enables decision makers in controlling and understanding the proposed algorithm classifiers more than that of MCAR. Lastly, we have evaluated a number of rule ranking parameters and their different combinations and found out that (CONFIDENCE-SUPPORT-CARDINALITY) usually leads to higher accuracy classifiers with smaller numbers of rules.

For MCAC algorithm, experimentations against two types of data (UCI and scheduling data sets) have been conducted to evaluate the effectiveness of the proposed algorithm in classifying test cases and producing multiple labels rules. The measures of evaluation are Label-Weight, Any-Label, First-Label, accuracy and number of rules and the contrasted algorithms are Rank-Label, MMAC for multi-label, and MCAR, CBA, PART, and C4.5 for single class. The results of the experiments can be summarised as follows:

1) The MCAC algorithm was able to produce multi-label rules with two, three and four class labels from both the UCI and the scheduling data sets. These rules correspond to important knowledge that both the classifier accuracy and the end-user benefited from.

2) MCAC outperformed the considered algorithms on the real world application data related to a scheduling problem with respect to the First-Label evaluation. Further, the Label-Weight and Any-Label results of the proposed algorithm are better than those of the Rank-Label and MMAC algorithms for the same data sets.

3) For the UCI data collection, the single label version of MCAC algorithm outperformed RIPPER, C4.5, PART, CBA and MCAR and the won-lost-tie is 13-7-0, 13-7-0, 14-6-0, 12-8-0 and 9-10-1 respectively with reference to the accuracy.

4) MCAC not only outperformed the other algorithms with respect to accuracy and other predictive based performances, but also generated smaller classifiers on the trainer scheduling data sets than the rest of the AC and rule based classification algorithms. This is because of the hyperhueristic behaviour in applying the local search methods while building the schedule.

Next chapter a crucial domain related to phishing classification is investigated and we show the applicability of our algorithms (MAC, MCAC) on data sets generated from such domain.

Chapter Six

Case Study: Website Phishing Identification

6.1 Introduction

Internet is equally important to individual, commercial and organisational users because of the online trading. Nevertheless, internet-users may be vulnerable to different types of web-threats that may cause financial damages, identity theft, loss of private information, brand reputation damage and loss of customer's confidence in e-commerce and online banking (Liu and Ye, 2001). Therefore, internet suitability for commercial transactions becomes doubtful.

Phishing is treated as a kind of web threat and is known as the skill of mimicking a website of a legitimate enterprise with the means to obtain confidential information such as national insurance numbers, passwords, and bank account numbers (Dhamija et al., 2006). Phishing websites are created by dishonest individuals to imitate genuine websites. These websites have high visual similarities to the legitimate ones in an attempt to defraud the honest internet-users. A report published by "Gartner" (Gartner, 2011), which is a research and advisory company shows that phishing attacks are increasing rapidly. Gartner estimated that theft through phishing attacks costs U.S. banks and credit card companies around $2.8 billion annually. In 2011, the Director of Cisco's security-technology-business-unit issued his concerns that today's main attacks focus on gaining access to corporate accounts that contain valuable financial information.

Phishing websites are expected to be more common in the future, thus smart solutions are needed to keep pace with the continuous evolution of this problem. The smart solutions are the subject of our interest in this chapter that can be combined with the heuristic-based approach. In fact, the accuracy of the heuristic-based solution mainly depends on a set of discriminative features extracted from the website. Hence, the way in which those features are processed plays an extensive role in accurately identifying websites, and therefore, an

effective intelligent based method when merged with the heuristic method can be essential for making a good decision.

Associative Classification (AC) is one of the promising approaches that can make use of the features extracted from the websites to find patterns among them (Costa, et al., 2013). This approach normally devises classifiers that are accurate so that the decision-making process becomes reliable simply because decisions are made based on rules discovered from historical data intelligently. Although plenty of applications offered for combating phishing websites, few of them make use of AC (Mohammad, et al., 2012; Abdelhamid, et al., 2013a).

Phishing is a typical classification problem (Abdelhamid, et al., 2013a) in which the goal is to assign each test data (new website) one of the predefined classes (phishy, legitimate, suspicious, etc). Precisely, once a website is loaded on the browser a set of feature values will be extracted from it. Those features have a strong influence in determining the type of the website by comparing them to rules that have been previously found by the AC algorithm from the historical data (former labelled websites). Then the chosen rule's class will be assigned to the browsed website and an appropriate action will take place. An example of website features includes "IP address, long URL, uses '@', https and SSL, age of domain, etc".

Since the AC approach has upsides over other traditional classification approaches as discussed earlier in Chapters 2, 3 and 4. In this chapter, the problem of phishing detection is investigated by showing the importance of this problem and current common solutions to it. Then, we evaluate AC primarily MCAC and MAC algorithms and compare them with other AC and rule induction algorithms on real data related to phishing. The data set used has been collected from Phishtank and Millermiles archives (Phishtank, 2006; Millersmiles , 2011), which are a free community site for sharing phishing data. In addition, the legitimate websites were collected from yahoo directory. The evaluation measures used in the comparison are accuracy, number of rules, Any-Label, and Label-Weight (Thabtah, , 2007).

We show that MCAC algorithm is able to extract rules representing correlations among website's features. These rules are then employed to guess the type of the website. The novelty lies in the new type of classifiers that contain rules associated with set of class

probabilities which are utilised in forecasting the type of the website. These class probabilities denote class weights per rule's set of classes which are computed by MCAC during rule learning step. This has improved the performance in regards to different evaluation measures as discussed in Section 6.5.

Chapter six is divided into different sections where Section 6.2 surveys common related technical and non-technical approaches to phishing detection besides phishing steps. Section 6.3 sheds the light on the features related to the problem of website phishing classification and shows the features that we select for the experimental results. Sections 6.4 and 6.5 are devoted for experimentations where we demonstrate, the data collection process, the evaluation measures, the compared algorithms, the results, and their analysis. Lastly, the chapter summary is given in Section 6.6.

6.2 Phishing Detection

In this section, we review common intelligent phishing detection approaches from the literature, after shedding the light on general steps required to solve the website phishing problem and its general combating approaches. Further, the section starts by showing the phishing life cycle.

6.2.1 Phishing Lifecycle

Social engineering which is the act of deceiving people to obtain sensitive information can be combined with computerised technical tricks in order to start a phishing attack (Aburrous,et al., 2010a). Figure 6.1 depicts the general steps conducted in the phishing life cycle. Phishing websites has become a serious problem not only because of the increased number of these websites but also the intelligent strategies used to design such websites, therefore even users having extensive experience and knowledge in computer security and internet might be deceived (Sanglerdsinlapachai and Rungsawang, 2010).

According to Figure 6.1, the phishing attack begins by sending an e-mail that seems to be from an authentic organisation to users urging them to change their data by selecting a link within an e-mail. E-mails remain a spreading channel for phishing links since 65% of phishing attacks start by visiting a link received within an e-mail (Kaspersky Lab, 2013). Other methods of distributing phishing URLs include, Black Hat search engine optimization (Black Hat SEO) (Seogod, 2011), Peer-to-peer file sharing, vulnerable websites such as blogs, forums, instant messaging (IM), and Internet Relay Chat (IRC)(Kirda and Kruegel, 2005).

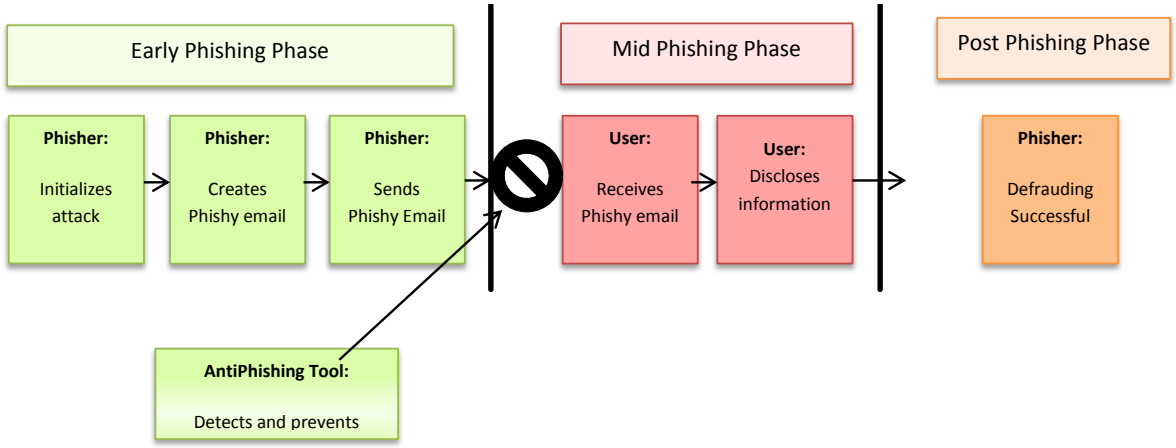

Fig. 6.1 Phishing life cycle

6.2.2 General Steps to Handle Phishing

The main steps that need to be addressed to solve the phishing problem are the following (Horng, et al., 2011):

1) Identification of required data: For any given problem we need a set of attributes, which are already measured or preset. These should have some influence on the desired output (classifier). Thus, a set of input and output attributes should be identified.
2) Training set formation**:** The training data set consists of pairs of input instances or examples and desired target attribute (class). There are many sources of phishing data such as Phishtank.
3) Determination of the input feature: The classifier accuracy depends on how the training examples are represented and how features have been carefully selected.

The feature selection process should discard irrelevant features as possible in order to minimise the training data set dimensionality, so the training process can be effectively executed. We show later in this chapter the ways we assessed the feature before choosing them.

4) Applying the classification algorithm: The choice of a mining algorithm is a critical step. There are wide ranges of mining methods available in the literature where each of these classification approaches has its own pros and cons. Three main elements in selecting a classification approach are a) The input data characteristics, b) the classifier predictive power measured by the accuracy, and c) the simplicity and understandability of the output. Overall, there is no single classifier that works best for all given data, and classifier performance largely relies on the training data set characteristics. For this step, we selected AC since it has many distinguishing features particularly the high predictive accuracy and the understandability of output derived.
5) Classifier evaluation: The last step is to test the derived classifier performance on test data.

6.2.3 Non-Technical Approaches to Minimise Phishing

The known general non-technical methods to combat phishing are (James, 2005):

- Legal solutions: Followed by many countries where the United States was the first to enact laws against phishing activities and many phishers have been arrested and sued. Phishing has been added to computer crime list in 2004 by Federal Trade Commission (FTC) which is a U.S government agency that aims to promote consumers protection (Kunz and Wilson, 2004). In the years 2005 and 2006, both the Australian and UK governments strengthened its legal arsenal against fraud by prohibiting the development of phishing websites and enacted jail penalties (http://www.finextra.com/news). Lastly, the Australian government also signed a partnership with Microsoft to teach the law enforcement officials how to combat different cyber-crimes (Government of Australia, 2011). Nevertheless, legal solutions do not sufficiently catch phishers since it is very difficult to trace phishers due to their quick disappearances in cyber world.

- Education: In combating phishing, consumer's education in order to raise awareness of this online crime is beneficial (Government of Australia, 2011). If internet-users are

convinced to inspect the security indicators within the website, the problem could substantially minimised. However, the important advantage for phishers to successfully trick internet-users is that the majority of internet-users lack basic knowledge of current online threats that may target them. Generally speaking, although raising awareness about phishing to online users may be seen as a promising direction, it is still a hard task to implement. This is because users are required to spend long time learning phishing methods, and phishers becoming more talented in creating new phishing techniques, which sometimes makes even security experts deceived.

6.2.4 Technical Approaches to Handle Phishing

Typically, the two most technical methods in fighting phishing are the blacklist and heuristic-based (Aaron and Manning, 2012; Sadeh, et al., 2007). In the blacklist method, the requested URL is compared with a predefined phishing URLs. The downside of the blacklist method is that it typically does not deal with all phishing websites as reviewing newly launched fake website takes a substantial amount of time before being added to the list. On the other hand, in the hueristic search approach several website features are collected and used to identify the type of the website. In contrast to the blacklist approach, the heuristic-based approach can recognise newly created fake websites in real-time (Miyamoto, et al., 2008). More details on these approaches are given in the next sub-sections.

Weaknesses that appeared when relying on abovementioned solutions led to the need to innovative solutions. Several solutions are offered these days to handle phishing such as MacAfee. Moreover, some non-profit organisations such as APWG (Aaron and Manning, 2012), Phishtank (Phishtank, 2006) and Millersmiles (Millersmiles , 2011) provide forums of opinions as well as distribution of the best practices against phishing from users' experiences. The success of an anti-phishing technique mainly depends on recognizing phishing websites. Although a number of anti-phishing solutions are developed, most of these solutions were unable to make highly accurate decisions causing a rise of false positive decisions, which means labelling a legitimate website as fake. We focus on technical solutions proposed by scholars in the literature in the following sub-sections for dealing with the website phishing problem.

6.2.4.1 Blacklist-Whitelist based Approach

A blacklist is a list of URL's that are thought to be malicious and have been collected using techniques such as user voting. So, whenever a website is launched, the browser refers to the blacklist to check if the launched website exists within the blacklist. If the check result is true, the browser warns the users not to submit any sensitive information. The blacklist could be saved either locally on the user's machine or on a server that is queried by the browser for every requested URL.

(Sheng, et al., 2009) showed that blacklists are usually updated at different frequencies. Precisely, it was estimated that 50% to 80% of phishy URL's are displayed in the blacklist 12 hours after their launch. Other blacklists such as Google's needs on average 7 hours to be updated (Dede, 2011). So it is necessary for a decent blacklist to be updated instantly to keep users safe from being victimised.

The blacklist approach has been deployed towards many solutions, one of which is Google Safe Browsing which uses a list of pre-defined phishy URLs to detect fake URLs. Another solution is Microsoft IE9 anti-phishing protection and SiteAdvisor (McAfee, 1997) which are basically database based solutions that are primarily created to catch malware attacks such as Spyware and Trojan horses. These contain an automated crawler that browses websites and builds threat ratings based on the visited URLs. Unlike blacklists or database based solutions, SiteAdvisor cannot identify newly created threats.

Another anti-phishing tool named VeriSign (Symantic, 2000) crawls millions of websites to recognise "clones" in order to distinguish phishing websites. One drawback with blacklists and crawling approaches might be that the anti-phishing parties will always be in a competition against the attackers. Netcraft (Netcraft Toolbar, 1995) is a small program that gets activated upon using a web browser. Netcraft relies on a blacklist which consists of fraudulent websites recognized by Netcraft and those submitted by the users and verified by Netcraft. Netcraft displays the location of the server where the webpage is hosted. This is helpful for experienced users of web hosting where for instance a webpage ending with ".ac.uk" is unlikely to be hosted outside the UK.

The opposite term of the blacklist is the whitelist, which is a set of trusted websites, while all other websites are considered untrusted. (Chen and Guo, 2006) proposed an Automated-Individual-Whitelist (AIWL), which is an anti-phishing tool based on an

Table 6.1 Phishing criteria from (Aburrous, et al., 2010b)

Feature Set	Phishing Feature Indicator
Domain Identity and URL	Via IP Address
	Require URL
	URL of Anchor
	DNS Details
	Strange URL
Encryption and Security	SSL Certificate
	Certification Authority
	Strange Cookie
	Distinguished Names Certificate(DN)
Java script and Source Code	Redirect Pages
	Straddling Attack
	Pharming Attack
	Using onMouseOver
	Server Form Handler
Contents and Page Style	Spelling Mistake
	Replicating a Website
	"Submit" Button
	via Pop-Up Windows
	Disabling Right-Click
Web Address Bar	Long URL Address
	Replacing Similar Characters for URL
	Adding Prefix or Suffix
	Using the '@' to Confuse
	Using Hexadecimal Character Codes
Social Human Factor	Much Stress on Security and Response
	Generic Welcome
	Buying Time to log on Accounts
	Buying Time to Access Accounts

individual user's whitelist of trusted websites. AIWL traces every login attempt made by the user through the utilisation of a Naive Bayes algorithm (Duda and Hart, 1973). In case a repeated successful login for a specific website is achieved, AIWL prompts the user to add the website to the whitelist.

One other solution that depends on the whitelist was presented in PhishZoo (Afroz and Greenstadt, 2011). PhishZoo builds profiles of trusted websites based on the fuzzy hashing technique. A website profile is a combination of several metrics that exclusively identifies that website. This approach combines whitelisting to recognise new phishing attacks with blacklisting and heuristic approach to warn users of attacks. The authors believed that phishing detection should be derived from the user's point of view since over 90% of users rely on the website appearance to verify its authenticity.

6.2.4.2 Fuzzy Rule based Approaches

One approach employed in (Aburrous,et al., 2010a) is based on experimentally contrasting few classification algorithms after collecting dissimilar features from a range of websites as displayed in Table 6.1. Those features varied amongst three uncertain set values

(Legitimate, Genuine, Doubtful). To evaluate the selected features the authors conducted experiments using the following algorithms in Weka, C4.5, PRISM, PART and RIPPER. The results uncovered a significant association between "Domain Identity" and "URL". However, no justifications on the way features have been assessed.

The authors of (Aburrous,et al., 2010b) have used a larger set of features to predict websites' type based on fuzzy logic. Although, their developed method gave promising results in accuracy, it was not mentioned how the features have been extracted from the website and specifically features related to human factors. Furthermore, the knowledge used were established based on human experience rather intelligent data mining techniques, which is one of the problems we aim to resolve in this chapter. Lastly, the authors classified the websites as very-legitimate, legitimate, suspicious, phishy or very-phishy, but they did not clarify what is the fine line that separates one class from another.

6.2.4.3 Machine Learning Approaches

The majority of methods developed to solve phishing in ML are based on support vector machine (SVM). SVM is a known ML technique that has been effectively used to solve classification problems (Song, 2009). Its popularity comes from the results it produced particularly from unstructured problems like text categorization (Joachims, 2001). An SVM in general can be seen as a hyper-plane that splits the objects (points) belonging to a class (positive objects) from those that do not belong to that class (negative objects). This split is implemented during the learning step where the hyper-plan is obtained to split the positive and negative objects with maximal margins. The margin denotes the space from hyper-plane to the closest positive and negative object.

A method based on SVM was proposed in (Pan and Ding, 2006) to discover unusual activities, i.e. phishing, in websites based on two variables. The first one is based on the company's name shown in the domain name, and the second one is called the "page categorizer", which denotes properties related to structural features (Abnormal URL, abnormal DNS record, etc) that are hard to be duplicated.

Six different structural features have been chosen, and Vapkin's SVM algorithm (Cortes and Vapnik, 1995) was used to determine whether the website is phishy or not. Tests on a limited data set consisting of 379 URLs revealed that the "Identity Extractor" is an

important feature related phishy URLs, and "page categorizer" feature correlates to the "Identity Extractor" features results. Overall, the accuracy derived by this method was 84%. A solution to increase this method's accuracy would be by employing additional features.

In (Sadeh, et al., 2007), the authors compared some commonly used machine-learning methods including SVM, decision trees, and Naïve Bayes on the problem of email phishing. A random forest algorithm called "Phishing Identification by Learning on Features of Email Received" (PILFER) was implemented. A data set consisting of 860 phishy emails and 695 legitimate were used in the experiments. The authors used a number of features for detecting phishing emails those are "IP based URL's, age of domain, non-matching URL's, having a link within the e-mail, HTML emails, number of links within the e-mail, number of domains appears within the e-mail, number of dot's within the links, containing JavaScript and spam filter output". The authors concluded that PILFER can be enhanced towards classifying emails by combining all the 10 features except "Spam filter output" with those shown in Table 6.2. PILFER has good accuracy in identifying phishing emails.

Table 6.2 Features added to PILFER to classify websites

Phishing Factor Indicator	**Feature Clarification**
Site in browser history	If a site not in the history list then it is expected to be phishing.
Redirected site	Forwarding users to new webpage.
tf-idf	Search key terms on a page then checks whether the current page is present in the result.

6.2.4.4 CANTINA based Approaches

A method proposed in (Guang et al., 2008) suggested utilising "Carnegie Mellon Anti-phishing and Network Analysis Tool" (CANTINA) (Zhang Y., et al., 2007). This content-based technique reveals the type of websites using the term-frequency-inverse-document-frequency (TF-IDF) measure (Song, 2009). TF-IDF assesses a document's word importance by counting its frequency. CANTINA calculates the TF-IDF for a given webpage then takes the five highest TF-IDF terms and adds them to the URL to find the lexical signature. Finally the lexical signature is fed into a search engine.

When the current webpage is among the first 30 results, it is considered a legitimate webpage. If not, it is phishy. If the search engine returns zero result, the website is labelled as phishy. To overcome the zero result, the authors combined TF-IDF with some other feature, e.g. (suspicious URL , Age of domain, dots in URL, etc). A limitation of this method is that some legitimate websites consist of images so using the TF-IDF may not be suitable. In addition, this approach does not deal with hidden texts, which might be effective in detecting the webpage type.

Another approach that utilises CANTINA with an additional attributes was proposed in (Sanglerdsinlapachai and Rungsawang, 2010) where a small data set having 100 phishy and 100 legitimate websites has been used. Eight features have been utilised as inputs to detect websites type (suspicious link , domain age, TF-IDF , suspicious URL, , IP address, dots in URL, known image forms). Some changes to the features have been performed during the experiments as follow:

1. The "Forms" feature is set as a filter to begin the process of deciding the legitimacy of the website since fraudulent sites that may cause the loss of users' information must have input blocks within the "Forms".
2. According to the authors, "Known image" and "Domain age" features are disregarded since they hold no significance.
3. The similarity between a fuzzy webpage and top-page of its domain is a newly suggested feature.

The authors have performed three types of experiments against their data set. The first experiment evaluated a reduced CANTINA feature set "IP address dots in URL, suspicious URL , IP address and suspicious link" and the second experiment tested whether the new feature "domain top-page similarity" plays a significant role in uncovering the website type. The third experiment evaluated the results after adding the new suggested feature to the reduced CANTINA features used in the first experiment. By comparing the performance after adding the new feature, a number of classification algorithms showed that the error rate results of the new feature played a key role in detecting the type of the website. Neural Network- Back Propagation algorithm (NN-BP) achieved the best accuracy with an error rate of 7.5% whereas Naïve Bayes derived the lowest performance with 22.5% error rate.

6.2.4.5 Image based Approaches

One promising approach proposed by (Liu, et al., 2005) detected the type of websites by comparing phishy and non-phishy sites based on visual similarity. This technique breaks down the webpage into block regions depending on "visual cues." The visual similarity between a fake webpage and a legitimate one is evaluated using three metrics: block level similarity; layout similarity, and overall style similarity based on the matching of the block regions. A webpage is considered phishy if any metric has a value higher than a predefined threshold. The authors collected 8 phishing webpages and 320 official bank sites, then carried out the experiment which revealed an acceptable error rate. The downside of this work is the small data set size.

Lastly, in (Dhamija and Tygar, 2005), a method called Dynamic Security Skins (DSS) was disseminated. Since the system designer and the phisher rely on the interface to protect or defraud users, this approach used an agreed discrete image that allows a remote server to prove its identity to the user for an easy verification. This technique requires users verification based on comparing the user's expected image with an image generated by the server. The authors implemented their method by developing an extension to Mozilla Firefox browser. The main drawback of this method is that the users bear the burden of deciding whether the website is phishy or not, thus users need to be conscious of the phishing and look for signs that the website he is visiting is in fact a spoof website. This approach also suggested a fundamental change on the web infrastructure for both servers and clients, so it can succeed only if the whole online industry supports it.

6.3 Website Features

6.3.1 Feature Preparation

There are several features that distinguish phishing websites from other ones in the research literature of phishing. In this section, we conduct websites features assessment based on frequency analysis for a number of features collected from the previous researches, i.e. (Miyamoto, et al., 2008; Mohammad, et al., 2012). These features contribute to the classification type of the websites. Particularly, a frequency analysis experiment that counts each feature using over 1350 websites collected from different sources. Phishing websites were collected from Millersmiles and Phishtank data archives, which are community sites for sharing phishing data. The legitimate websites were collected from yahoo directory

using a web script developed in PHP. The script was plugged with a browser and we collected 601 legitimate and 752 phishing websites. The aim for this experiment is to select scientifically the common features that may help in assessing the determination of the website's type accurately.

In our study, sixteen different features plus the class have been identified after performing the frequency analysis against the different phishy URLs collected. The result of the analysis is depicted in Table 6.3 where we can see each feature and its associated frequency rate computed from the gathered data set. For example, "Age of Domain" and "Request URL" (Explained shortly in Section 6.3.2) are common features since they constitutes high rate appearing in very large portion of the websites. Nevertheless, "URL having the @ symbol" constitutes 6.8% which relatively low rate but indeed always associated with phishy websites, and thus it has high impact on distinguishing this type of websites.

The chosen feature shown in Table 6.3 taking either a binary or a ternary values where binary features hold either "phishy" or "legitimate" because the existence or lack of the

Table 6.3 The selected features set

Website Feature	**Percentage rate**
IP address	20.5%
Long URL	51.0%
URL's having @ symbol	6.8%
Prefix and suffix	25.4%
Sub-domain (dots)	42.8%
Fake of HTTPs protocol SFH Handler	89.2%
Request URL	100%
Server Form Handler	5.7%
URL of Anchor	22.3$
Abnormal URL	20.5%
Using Pop-up Window	14.3%
Redirect Page	11.0%
DNS record	7.6%
Hiding the links	21.0%
Website Traffic	93.2%
Age of Domain	97.4%

feature within the website determines the value assigned to it. On the other hand, and for the ternary value features one more value has been added, i.e. “Suspicious”. For ternary value features, the existence of the feature in a specific ratio determines the value assigned for that feature. Later in the experimental section, we utilise Chi-Square testing to further assess the selected features set.

6.3.2 The Selected Features

In this subsection we explain the features that have been used for experimentations and their corresponding rules.

1. **IP address:** Using an IP address in the domain name of the URL is an indicator someone is trying to access the personal information. An IP address is like http://91.121.10.211/~chems/webscr/verify Sometimes the IP address is transformed to hexadecimal like http://0x58.0xCC.0xCA.0x62.

Rule: If IP address exists in URL → Phishy **else** → Legit

2. **Long URL:**Phishers hide the suspicious part of the URL to redirect the information’s submitted by the users or redirect the uploaded page to a suspicious domain. Scientifically, there is no standard reliable length that differentiates between phishing URLs and legitimate ones. (Mohammad, et al., 2012) suggested when the URL length is greater than 54 characters the URL can be considered phishy.

Rule: If URL length < 54 → Legit URL length ≥ 54 and ≤ 75 → Suspicious else → Phishy

3. **URL’s having @ symbol:** One of the elements that may cause suspicion in a URL is the “@” symbol. The “@” symbol leads the browser to ignore everything prior it and redirects the user to the link typed after it.

Rule: If URL has ‘@’ → Phishy else Legit

4. **Adding prefix and suffix:** Phishers try to scam users by reshaping the suspicious URL so it looks legitimate. One technique used to do so is adding prefix or suffix to the legitimate URL thus the user may not notice any difference.

Rule: If domain part has ' – ' → Phishy else → Legit

5. **Sub-domains:** Another technique used by the phishers to scam users is by adding a subdomain to the URL so users may believe they are dealing with an authentic website. An example: http://www.paypal.it.ascendancetheatrearts.co.uk

Rule: If dots in domain < 3 → Legit else if = 3 → Suspicious else → Phishy

6. **Fake HTTPs protocol/SSL Final:** The existence of HTTPs protocol every time sensitive information is being transferred reflects that the user certainly connected with an honest website. However, phishers may use a fake HTTPs protocol so that the users may be deceived. So checking that the HTTPs protocol is offered by a trusted issuer such as GeoTrust, GoDaddy, Thawte, VeriSign, etc, is recommended.

Rule: use of https & trusted issuer & age ≥ 2 years → Legit using https & issuer is not trusted → Suspicious else → Phishy

7. **Request URL:** A webpage usually consists of text and some objects such as images and videos. Typically, these objects are loaded into the webpage from the same server of the webpage. If the objects are loaded from a domain other than the one typed in the URL address bar, the webpage is potentially suspicious.

Rule: request URL % <22% → Legit request URL % ≥ 22% and < 61% → Suspicious else → Phishy

8. **URL of Anchor:** Similar to URL feature but here the links within the webpage may point to a domain different from the domain typed in the URL address bar.

Rule: URL anchor % <31% → Legit URL anchor % ≥ and ≤ 67% → Suspicious else → Phishy

9. **Server Form Handler (SFH):** Once the user submitted his information; the webpage will transfer the information to a server so that it can process it. Normally, the information is processed from the same domain where the webpage is being loaded. Phishers resort to make the server form handler either empty or the information is transferred to somewhere different than the legitimate domain.

Rule: SFH If ' about:blank' or empty → Phishy
SHD redirects to different domain → Suspicious
else → Legit

10. **Abnormal URL:** If the website identity does not match a record in the WHOIS database (WHOIS 2011) the website is classified as phishy.

Rule: No hostname in URL → Phishy
else → Legit

11. **Using Pop-up Window:** Usually authenticated sites do not ask users to submit their credentials via a popup window.

Rule: rightClick disabled → Phishy
rightClick showing alert → Suspicious
else → Legit

12. **Redirect page:** When a user clicks on a link may be unaware that he's redirected to suspicious webpage. Redirection is commonly used by phishers to hide the real link and lures the users to submit their information to a fake site.
13. **DNS record:** A[illegible]site is classified phishy. Phishers aim to [illegible]ssible since the phishing webpage often lasts for short period of time and the URL is not valid any more. DNS record provides information about the domain that is still a live at the moment, while the deleted domains are not available on the DNS record.

Rule: redirect page #s ≤ 1 → legit
redirect page #s >1 and < 4 → Suspicious
else → phishy

Rule: No DNS record → Phishy
else → Legit

14. **Hiding the links:** Phishers often hide the suspicious link by showing a fake link on the status bar of the browser or by hiding the status bar itself. This can be achieved by tracking the mouse cursor and once the user arrives to the suspicious link the status bar content changed.

Rule: change of status bar onMouseOver → Phishy
no Change → Suspicious
else → Legit

15. **Website Traffic:** Legitimate websites usually have high traffic since they are being visited regularly. Since phishing websites normally have a relatively short life; they

Rule: webTraffic <150,000 → Legit
webTraffic >150,000 → Suspicious
else → Phishy

have no web traffic or they have low ranking. It has been recommended that a legitimate webpage has a rank more than or equal to 150,000 in the Alexadatabase (Alexa, 2011).

16. **Age of Domain:** Websites that have an online presence of less than 1 year, can be considered risky.

Rule: age ≥ 6 months → Legit
else → Phishy

6.4 Applying MAC and MCAC to Phishing Website

The process of detecting the type of website is a classification problem where different features are utilised to learn important hidden knowledge among them. This set of knowledge is in fact the classification system that in turn is used to automatically guess the type of website when a user browses it. We have identified different features discussed earlier related to legitimate and phishy websites and collected over 1350 different websites from difference sources as discussed in Section 6.1. The sample of the phishing data (8 examples) for some features is shown in Table 6.4. Some of the collected features hold categorical values those are “Legitimate”, ”Suspicious” and “Phishy”, these values have been mapped with numerical values 1,0 and -1 respectively.

Normally, there are two classes where a website can be classified into: Legitimate or phishy. Though, one of the proposed AC algorithm in this thesis can discover not only rules associated with one class but also a set of classes, i.e. (legitimate or phishy). MCAC

algorithm can produce a new type of rules based on a new class label not previously seen in the training data set which we name "Suspicious". When a website is considered suspicious that means it can be either phishy or legitimate and based on the computed probabilities assigned to the test data by the classifier rule, and the end-user can make a more accurate decision.

Different types of experiments (Section 6.5) have been conducted to evaluate the performance of both MAC and MCAC on the phishing data that we have collected. Particularly, we produced the accuracy, classifier size for all considered algorithms against the phishing data set. Furthermore, the new type of knowledge generated by MCAC has also been investigated by comparing MCAC with MMAC using Label-Weight and Any-Label measures (defined earlier in Chapter 5). Finally, we have conducted experiments on a reduced features set of the phishing data to assess the change in the considered algorithms' performance with respect to accuracy. All these experiments along with deep analysis on their results are discussed in the next section.

The main criteria used for the algorithms results evaluation are:

1) Classifiers one-error rate (%) or accuracy
2) Classifier size (# of rules)
3) Multi-label rules generation by MCAC
4) Reduced features set assessment and its impact on accuracy

Table 6.4 Sample data for the phishing problem using a number of features

URL anchor	**Request URL**	**SFH**	**URL length**	**Having '@'**	**Prefix suffix**	**IP**	**Sub domain**	**Web traffic**	**DNS record**	**Class**
0	-1	0	-1	1	1	1	-1	0	1	1
1	1	0	0	1	1	1	0	-1	1	1
1	1	1	0	-1	1	1	0	1	1	-1
0	0	1	-1	1	1	1	1	-1	1	1
-1	-1	0	0	1	1	-1	0	0	-1	-1
1	1	1	0	1	1	1	1	-1	1	1
1	1	1	1	-1	-1	1	1	1	-1	-1
1	1	1	1	1	1	-1	1	1	1	-1

6.5 Experimental Results

6.5.1 Settings

Different AC and rule based algorithms have been used to evaluate the performance and applicability of MAC and MCAC algorithms on the data set collected. The main algorithms used in the experiments beside ours are (CBA, MCAR and MMAC) from AC community, and (C4.5, PART, RIPPER) for rule induction and decision trees. The selection of these algorithms is based on the fact that they are rule based and they use different learning methodologies for fair comparison.

Experimentations have been carried out on an I3 with 2.3 Ghz. The *minsupp* and *minconf* thresholds have been set to 2% and 50% respectively in the experiments as in Chapter 5 for CBA, MMAC, MCAR and our algorithms.

6.5.2 Results Analysis

We start the result section by measuring the predictive power of the MAC algorithm by comparing its classifiers resulted from the experiments with those of the contrasted algorithms. Figure 6.2 summarises the prediction accuracy (%) produced by the considered algorithms for the phishing problem data set. It is obvious from the graph that MAC outperformed the other AC algorithms and the rule based ones in predicting the type of the websites. In particular, MAC outperformed RIPPER, C4.5, PART, CBA, and MCAR with 1.86%, 1.24%, 4.46%, 2.56%, 0.8% respectively. Overall, the prediction accuracy obtained from all algorithms is acceptable and that reflects features goodness in predicting the website class. One main reason for achieving higher predictive accuracy by the AC algorithms is their ability of discovering data insights that other rule based classification are unable to detect. In addition, MAC employs an accurate prediction method that takes into account group of rules decision rather than a single rule decision which makes the classification process more accurate.

Figure 6.3 displays the number of rules generated by all algorithms against the considered data set. The figure stresses that MCAR generates the largest number of rules followed by MAC and CBA if contrasted to decision trees, rule induction or hybrid classification (PART). The main cause of the larger classifiers of the AC algorithms is because a) the features in the data set is highly correlated and b) AC algorithms allow a training case to be used more than once in learning the rules unlike traditional classification

algorithms that only allow each training case to be used only once for a particular rule, which explains its smaller size classifiers. Though, with rule pruning many redundant rules have been removed during building the classifier by the AC algorithms.

To investigate the performance of MCAC algorithm especially the class probabilities role in the generated rules from the phishing data, Figure 6.4 compares MCAC and MMAC algorithms using two evaluation measures named Any-Label and Label-Weight, which have been discussed in Chapter 5. This figure shows that MCAC outperformed the MMAC algorithm in both Label-Weight and Any-Label measures on the phishing data. One possible reason for the increase of both evaluation measures for the MCAC algorithm is due to the new extracted knowledge by the MCAC algorithm that represent rules connected with a new class (suspicious) which enable end-users identify suspicious websites. In fact, the rule discovery method of the proposed algorithm extracts the multi-label rules early without the need to perform recursive learning which necessitates learning from independent sets of the training data similar to covering approaches. Instead, the MCAC algorithm learns the multi-label rules from the whole training data set once by discovering single label rules that survive *minsupp* and *minconf* early and merge only those that share antecedent (body) to generate the multi-label rules.

Another possible contributor to MCAC's good performance in accuracy is its ability to reduce the default class usage during the prediction step in which if no single rule is applicable to the test case, the prediction procedure of the MCAC algorithm takes on the group of rules that partly matching the test data and assigns the largest group class to the test case. Finally, rather than classifying websites that are neither phishy nor legitimate to phishy class the new rules discovered by MCAC algorithm are able to cover these websites by their class probabilities. This reduces the number of misclassifications on test data and improves the predictive performance of the classifiers. So, questions such as "is the website close to the phishy or legitimate class?" and "by how much?" are answered by MCAC's classifier.

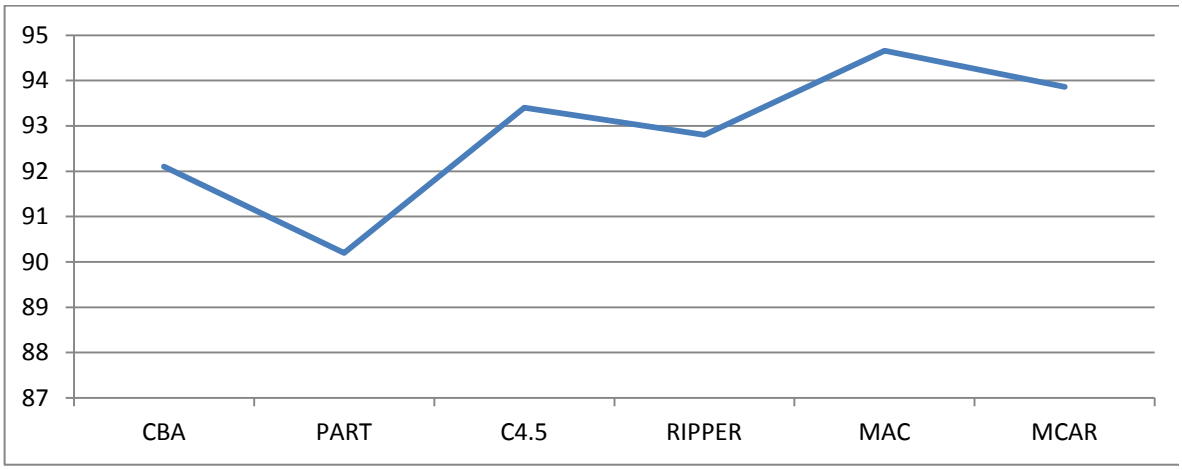

Figure 6.2 The classification accuracy (%) for the contrasted algorithms derived from the phishing data

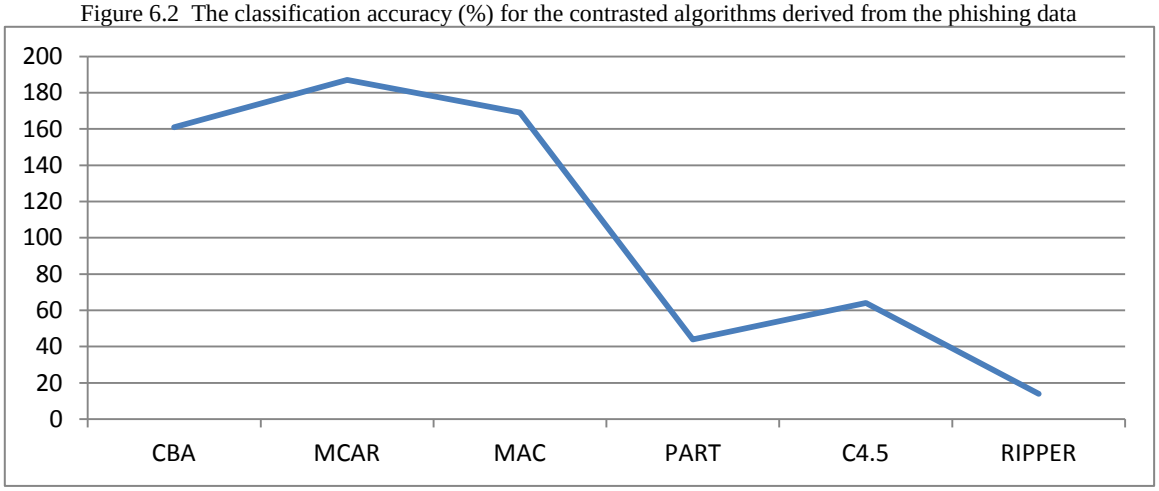

Figure 6.3 Average number of rules generated by the contrasted algorithms derived from the phishing data problem

To signify the importance of the additional knowledge produced by the proposed algorithm Figure 6.5 displays the number of rules with respect to their consequent part (class labels on the right hand side). The proposed algorithm was able to extract rules from the phishing data set that are connected with new class (phishy or legitimate) to deal with test examples that are neither fully phishy or legitimate. This is accomplished using computed probabilities associated with the class labels in the discovered rules. In particular, Figure 6.5 shows that the MCAC algorithm generated 24 rules that represent "Legitimate Or Phishy" class. These rules are linked to websites that are suspicious that most current algorithms classify them to "Phishy" class. In other words, the MCAC algorithm was able to extract rules that current AC algorithms and traditional classification algorithm ignore.

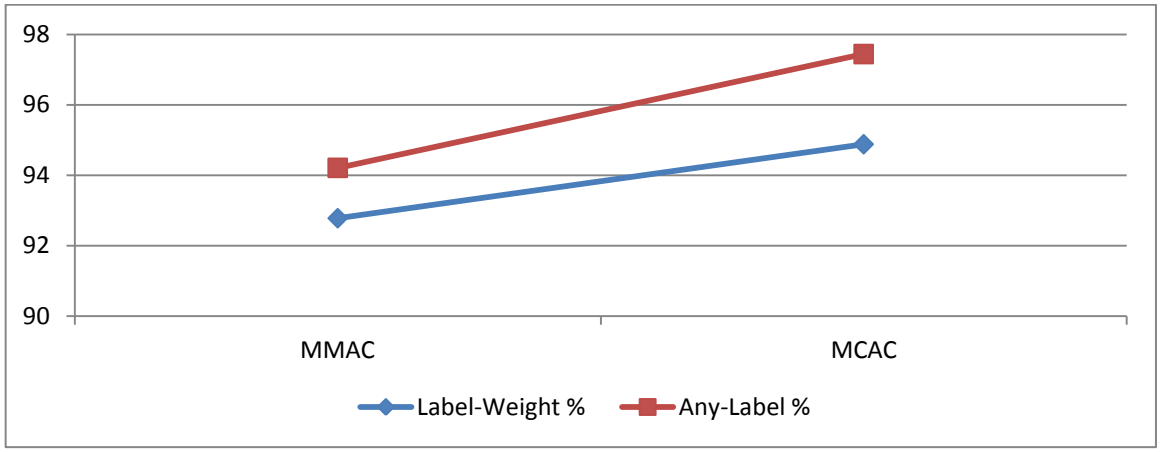

Figure 6.4 Label-Weight and Any-Label measures (%) for MCAC and MMAC algorithms

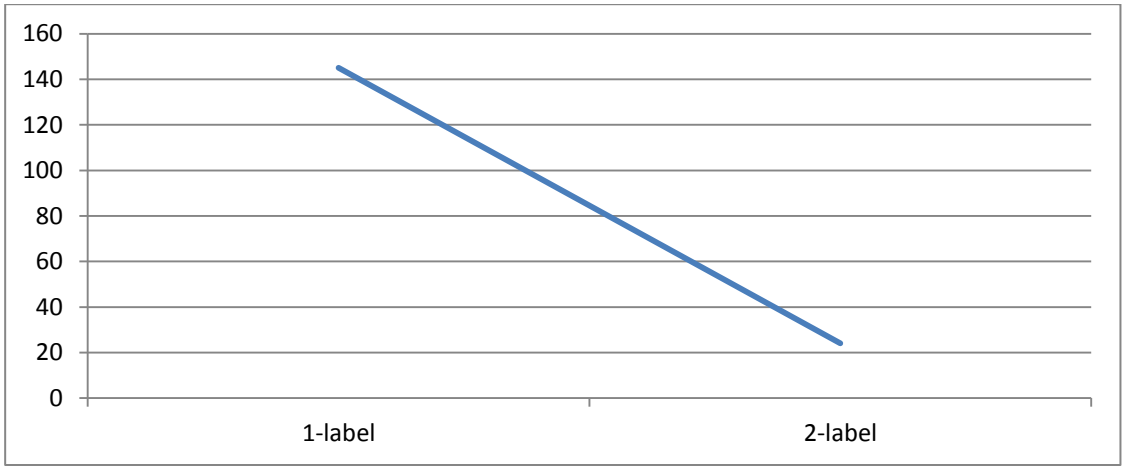

Figure 6.5 Number of class labels per rule derived by the MCAC algorithm from the phishing data

6.5.3 Reduced Features Results

We have reduced the number of features in order to identify the smallest significant ones that are able to guess the type of the website. In addition, selecting small set of features may eliminate the noise in choosing features, which occurs whenever there are irrelevant features presented within the training data set, which in turn causes an increase in the classification errors.

We have applied the Chi-Square feature selection measure (Witten and Frank, 2002) to further reduce the selected different features that we have collected. The aim of this assessment is to end up with a smaller set of features that signify the classification of phishing websites. We have employed the Chi-Square filter in Weka (Weka, 2011) as a

feature selection criterion to accomplish the above task and Table 6.5 depicts the results of the top features that have high significance based on Chi-square feature selection. A full detail of the rune is given in Appendix A (Figure 19). Chi-square evaluates the relevancy of variables for classification problems. It is a known data hypothesis method from statistics, which evaluates the correlation between two variables and determines whether they are correlated. For our data set, each feature correlation with the class attribute has been evaluated. The test for correlation when applied to a population of objects determines whether they are positively correlated or not.

Chi-Square has been employed in many practical domains for feature selection, e.g. (Li, et al., 2001; Ramaswami and Bhaskaran, 2009; Thabtah, et al., 2009), to assess the relevancy of the attributes in a classification data set. As mentioned above, the evaluation

Rank	Chi-Square	Feature
1	744.1558	SFH
2	667.5554	on_mouseover
3	406.6605	SSLfinal_State
4	358.5917	popUpWidnow
5	198.1662	Request_URL
6	158.8848	Redirect
7	127.493	URL_of_Anchor
8	102.1857	Web_traffic
9	95.6012	URL_Length

of the sixteen website features using Chi-Square filter in Weka showed that nine features have correlation with the class attribute values and therefore they may impact on the process of phishing detection.

We consider the accuracy of the same classification algorithms used previously against the reduced features set of the phishing problem data. Figure 6.6 displays the classification accuracy on the reduced number of features of all single label algorithms considered. We noticed that the classification accuracy was not heavily impacted on average for all classification algorithms used. The accuracy has been reduced on average only by 0.6% if contrasted with that derived from all features set (Section 6.5.2). This reflects the goodness

of the reduced features set in classifying the type of the websites. However, for MAC the classification rate has slightly increased.

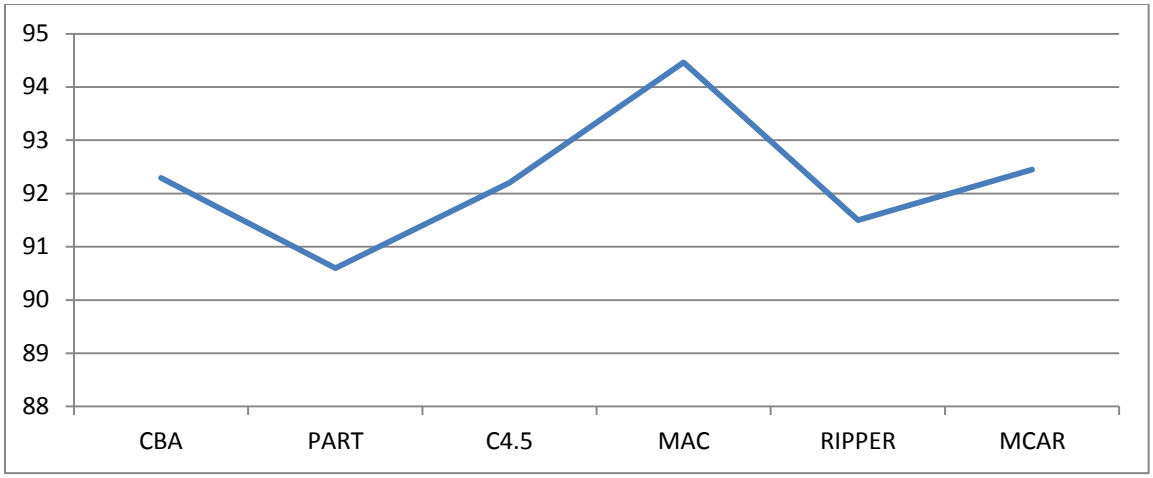

Figure 6.6 The accuracy (%) for the contrasted algorithms derived from the reduced features set of the phishing data set

6.6 Chapter Summary

In this chapter, the website phishing classification problem has been investigated in the context of AC. To be exact, we defined the website phishing as a classification problem, and surveyed its common intelligent approaches in data mining and ML. Moreover, the different significant features related to identifying the type of website have been explained along with the data collection method and the sources of the data sets. Lastly, the applicability of our algorithms on the website phishing classification data has been conducted. We compared our algorithms with other known rule based classification algorithms in two scenarios:

1) Sixteen different features data set
2) Reduced features data set based on Chi-Square feature selection measure

The Experimentation's aim is to measure how effective MAC and MCAC in classifying websites and producing multiple label knowledge. Label-Weight, accuracy, Any-Label, and the number of rules are the measures of evaluation and the contrasted algorithms are CBA, MCAR, MMAC, PART, C4.5 and RIPPER.

The results of the experiments are summarized below:

5) MAC outperformed the other algorithms on detecting phishing with respect to predictive power. Moreover, for MCAC, the Label-Weight and Any-Label results are better than those of MMAC on the same phishing data.

6) MCAC was able to produce multi-label rules from the phishing data generating rules associated with a new class label called “Suspicious” that was not in the training data set, which improved its predictive performance.

7) For the reduced features set experiments, the classification performance of MAC slightly enhanced since we have identified a smaller effective feature set for detecting the type of the website after applying Chi-Square feature selection method. The results of all considered algorithms have been consistent in detecting the phishing website.

In the next chapter, the thesis conclusions and possible future research directions are discussed.

Chapter Seven

Conclusions and Future Work

7.1 Research Summary

This thesis investigated several issues related to AC in data mining such as the extraction of new type of knowledge (rules) connected with multiple class labels. In addition, different enhancements of classifiers performance including rule pruning, rule sorting and test data class assignment steps, have been investigated. The results are two new AC algorithms named MAC and MCAC. MAC algorithm improved on AC steps: rule filtering, frequent ruleitem discovery, and test data prediction as we discuss in the next sub-sections. Whereas, MCAC algorithm contains a novel rule learning method that finds and generates multi-label rules early without recursive learning from data sets associated with one class. These two algorithms have been disseminated as shown in "Publication page" (Section IV). Lastly, the proposed algorithms have been evaluated on large numbers of data sets from UCI, website classification, and scheduling applications.

7.2 Critical Review and Research Contributions

In this section, the different contributions for the raised issues of Chapter 1 are summarised along with critically revealing the summary of the experimental results for each contribution.

7.2.1 Multi-Label Rules Discovery and Generation

The majority of the current AC algorithms discover only the largest frequency class linked with an attribute value in the training data set leading them to only derive single label rules. In other words, they are unable to find and extract all other classes connected with the attribute values. Nevertheless, finding the set of classes connected to the rule's body in the classifier is beneficial for both the classifier and the decision maker as stated early in Chapters 1 and 4.

Overall, most of the current AC algorithms miss the second, third, fourth, etc, class labels connected with a rule even when these classes have adequate representation in the training data and survived the *minsupp* and *minconf* thresholds. This may cause ignoring crucial knowledge. In Chapter 4, we proposed the MCAC algorithm to devise classifiers containing rules with multi-labels. Our algorithm finds the set of classes connected with a rule from the whole training data set at once and merges them in a disjunctive manner in the rule consequent based on their weight. Meaning, MCAC discovers and generates all class labels connected with a rule after surviving the *minsupp* and *minconf* requirements giving the domain expert valuable information that he might use in making key business decisions. Further, the classifier accuracy rate has enhanced since now a rule has multiple options (classes) when it comes to test data classification.

Once all the multi-label rules are formed during the learning phase each class associated with a rule gets assigned a weight/probability that corresponds to its actual representation. These classes are sorted within the multi-label rule based on their probabilities in a descending manner. Further, for each formed multi-label rule, a new support and confidence values are assigned to it based on the average support and confidence values for its entire corresponding classes. This surely gives the new multi-label rules the true ranking position in the classifier so laterally they can be used for predicting the test data classes. When a test data is about to classify, MCAC algorithm assigns the class probability of the rule rather the class itself especially when there are multiple classes in the rule's consequent.

Experimentations using numbers of data sets collected from UCI data repository and real application domains showed that MCAC algorithm can devise rules where each of which may be connected with two, three, four and even five class labels. These new rules have enhanced the classification accuracy of MCAC classifiers if contrasted with those derived from MMAC, CBA and other AC and rule based classification algorithms. Furthermore, and for the UCI data sets, the MCAC algorithm first ranked class outperformed RIPPER, C4.5, PART, CBA, and MCAR and the won-lost-ties record are 16-4-0, 13-7-0, 15-5-0, 12-8-0 and 11-9-1 respectively with reference to classification accuracy. More details on the results are given in the experimental sections of Chapter 5.

7.2.2 Improving Classifiers Performance

In this thesis, four enhancements related to AC classification have been implemented, the first three are described below and the fourth one in Section 7.2.3:

A. Reducing the number of rules without impacting the accuracy rate of the classifier: For this issue, a new rule filtering method that checks each candidate rule against the training data set and saves rules that have high data coverage in the classifier is proposed. Our rule filtering method reduces the overfitting on the training data set by examining the applicability of the candidate rules on the training data set without considering class similarity as current AC algorithms do. Experimental results using different data sets have revealed that MAC classifier is smaller than that of other AC algorithms like MCAR on the majority of the classification data sets which we consider.

B. Enhancing the class assignment process by proposing a group of rules prediction procedure: We proposed a class assignment procedure(s) in MAC and another in MCAC. The first proposed procedure ensures that all rules relevant to the test data are used in class allocation process to minimise biased decisions. During class assignment, our procedure divides all relevant rules to the test data into groups and counts each group's number of rules. Then, it allocates the test data the class that belongs to the group having the largest number of rules. For the MCAC algorithm, we consider first a single rule matching all test data attributes value. If this condition is true then the rule class is allocated to the test data, otherwise the former procedure described above is invoked instead. MCAC's class assignment procedure takes advantage of both one-rule and multiple-rule classification approaches and therefore it can be considered a hybrid procedure. Experimentations in Chapter 5 depicted that the proposed class assignment procedures reduce the use of the default class and therefore positively affected the accuracy rate of the classifiers on large number s of data sets.

C. Minimising the number of TIDs intersections in the step of frequent ruleitems discovery: We present an enhancement over AC algorithms in the frequent

ruleitems discovery step. In particular, MAC only intersects the TIDs of the candidate ruleitems with identical class during any iteration and therefore massive numbers of unnecessary intersections are not performed which consequently improved this step. The frequent ruleitems discovery method proposed in MAC discovers all candidate rules by employing fast intersections among frequent ruleitems TIDs having similar class labels to produce the candidate ruleitems. This has led to huge saving in cutting down unnecessary intersections as discussed in the experimental results of Chapter 5. The results showed a reduction in the number of intersections in iteration 2 and its successors was almost 80% if compared with current vertical AC algorithms on the data sets we consider.

7.2.3 Rule Ranking Evaluation

Ranking rules before constructing the classifier is crucial in AC since the algorithm needs a way of discriminating among rules during classifying test data. Therefore, one can consider rule ranking a vital step since often higher rank rules get chosen first to be inserted into the classifier and lower rank rules are removed. There are few parameters used to favour rules mainly rule's confidence, support, and antecedent length. In this thesis, we have investigated different rule ranking formulas constituting the above mentioned parameters to seek the impact of this step on a) classification accuracy of the classifier and b) the number of candidate rules produced.

Experimental results depicted that rule's confidence is the most crucial criterion in rule ranking and had the largest impact on the classifier's accuracy, followed by rule's support, rule's length and finally rule's class distribution. We have considered a new parameter called "minority class distribution" for MAC algorithm to further discriminate among rules having identical confidence, support and length. The results on a number of UCI data sets revealed that on average the classifiers accuracy rate has enhanced by 1.56%. This shows the critical impact of our rule ranking method.

7.2.4 Detecting Phishing Websites

The problem of website phishing classification has been extensively investigated in Chapter 6. In particular, we highlighted the problem, the phishing lifecycle, non-technical

and technical solutions. The focus was on technical solutions related to data mining and ML which normally involve learning rules from the websites features.

After surveying large number of research articles related to phishing, we identified the common non-human based features (automatically extracting based features) by software tools related to phisning. The focus is based on automatically identifying the phishing activities so we limit the website features to those that are non-human based that can be extracted on the fly for the mining method.

We have collected over 1350 phishy and legitimate websites from different sources using an online script. Then we assessed the collected large set of features using frequency analysis and Chi-square testing feature selection methods to identify the minimal set of features that can assist in identifying website types. Finally, comprehensive experiments have been conducted using our algorithms (MAC, MCAC) and popular AC and rule based algorithms and with respect to various performance measures.

The results showed that MCAC was able to find new type of knowledge useful for the end-user that could also improve its performance in Label-Weight and Any-Label evaluation measures. Further, even the single version of MCAC (First ranked class classifiers) have achieved better accuracy rate if compared with other AC algorithms. For MAC, its classifiers predictive power was the best if contrasted with those of CBA, MCAR, RIPPER, C4.5 and PART. Finally, even on the reduced features set our algorithms consistently outperformed the other AC algorithms with respect to the performance measures used.

7.2.5 Testing the Proposed Algorithm on Real Data

The applicability of the proposed algorithms (MAC, MCAC) has been investigated on two types of data collections (UCI (20 data sets) and the trainer timetabling (6 data sets)) besides the case study on phishing. For the three types of data collections, we compared the proposed algorithms with a number of AC and rule based classification algorithms and using various evaluation measures. Experimental results on the above described data sets showed that the proposed algorithms scale well when contrasted with known AC algorithms like CBA, MCAR, MMAC and rule based algorithms like PART, C4.5 and RIPPER. In fact, the classification performance of MAC and MCAC on the data sets showed consistency in outperforming known AC and rule based algorithms with respect to

accuracy rate, Label-Weight, Any-Label and other evaluation measures. More details are given in the experimental sections of Chapters 5.

7.3 Future Work

7.3.1 Immune Systems based AC

One of the effective learning approaches that has been originated from the Natural Immune System (NIS) and have successfully applied in optimization, online security and data mining is Artificial Immune System (AIS). As a matter fact, AIS has been utilised in classification problem in last decade and devised a competitive performance results in accuracy rate. Examples of known classification algorithms that are based on AIS are clonal selection (Greensmith, et al., 2005) and negative selection (Do, et al., 2009). We believe that AIS can be used in AC especially to minimise the search space for rules by reducing the number of candidate rules. Hereunder, two attempts in using AIS within AC have been outlined.

There have been some initial attempts to adapt the learning methodology of NIS especially the clonal selection in AC context that have resulted in an algorithm named artificial immune system-associative classification (AIS-AC) (Do et al., 2009). The AIS-AC algorithm was proposed in 2005 and extended in 2009 and follows the evolutionary process by reducing the search space of the candidate rules by keeping just high predictive rules. This process is accomplished by extracting frequent 1-ruleitems after passing over the initial training data set, and generating the possible candidate ruleitems at iteration *N* from results derived at iteration *N* - 1 and so forth. The *minsupp* and *minconf* are utilised as sharp lines to discriminate among rulesitems at each iteration. Further, two new parameters are introduced named Clonal_rate and Max_generation. The clonal_rate (defined below) denotes the rate at which items in the candidate rules at given generation are extended, and it is proportional to the rule confidence.

$$Clonal\_rate = \frac{n.Clonal\_rate}{\sum_{i=1}^{n} conf(r_i)} \qquad (7.1)$$

where *n* is the number of rules at the current iteration, and the clonal_rate is a predefined user parameter. Once the candidate rules are extracted, they are tested on the training data

keeping only those that have one or more training example(s) coverage. The algorithm terminates once the complete training data set is covered or the Max_generation condition has been met (often set to 10). The candidate rules that have training data coverage are kept in the classifier. The AIS-AC algorithm applies the rules in the classifier on the test data similar to CBA prediction method.

Recently, another AIS based on AC called AC-CS was proposed in (Elsayed, et al., 2012). This algorithm follows the same track of the previously described AIS-AC and it uses the same strategies in deriving the rules and classifying test data. One simple difference between AC-CS and AIS-AC is that AC-CS builds the candidate rules in generations per class rather than at once and then merges each class rules set before evaluating the complete set of rules on the training data to determine the classifier.

Empirical evaluations using a limited number of UCI data sets indicated that the AIS proposed algorithms in (Elsayed, et al., 2012; Do et al., 2009) are highly competitive in accuracy and execution time to the "Predictive Apriori" algorithm (Weka, 2011) which is a simplified version of CBA that primarily uses Apriori algorithm for extracting the rules without pruning.

7.3.2 Test Data Training

Lazy AC as an approach was originated to maximize the predictive power of classifiers by minimising rule filtering to only candidate rules that wrongly cover training data while building the classifier, i.e. (Baralis, et al., 2008). Recently, (Veloso, et al., 2011) have proposed a new lazy approach in AC mining that primarily depends on the test data attribute values in reducing the rules set applicable to the test data in the classification step. Hereunder, we briefly shed the light on two different lazy learning methodologies and introduce an important issue in classification related to delaying learning rules until the classification step. This can be seen as a possible research starting point to minimise the search space for candidate rules.

The first learning methodology in lazy AC focuses on minimizing candidate rules filtering process aiming to accomplish high performance classifiers in regards to accuracy rate. Precisely, lazy AC algorithms that follow this methodology like L^3 (Baralis, et al., 2004) and L^3G (Baralis, et al., 2008) discard only candidate rules that have wrong classification when evaluating rules in the process of building the classifier. Meaning,

while evaluating rules on the training data to choose the best ones, all rules that either a) have correct classification on the training data or b) have not covered any training data, are stored in the classifier. The rules that correctly cover at least one training data are stored in a primary storage, and the rules that have no training data coverage are kept in a lower secondary storage. These two storages together are simply the classifier. Now, when the classification process of test data starts, rules in the primary storage are checked and when none of them is able to classify the test data rules in the secondary storage are utilised instead of the default class rule. This approach normally produces very large classifiers which may restrict its utilisation for applications.

A different approach was proposed in (Veloso, et al., 2007) which allows both training and testing examples to play a role in assigning the class to the test data. This learning methodology claims that deriving all candidate rules in the training phase could be problematic in cases when the *minsupp* is set to low values. Thus, suggesting using the test data attribute values as valuable information to reduce the search space of applicable rules. Meaning, attribute values in the training data that are similar to the test data attribute values are the only one used to learn the rules that in turn are used to assign the right class to the test data delaying the rules reasoning and merging it with the classification step. This according to (Veloso, et al., 2007) reduces the dimensionality of the training data though it requires learning from part of the training data and for each test data repeatedly similar to Naïve bayes even if data are not partitioned with respect to class labels as in Naïve bayes algorithm.

7.3.3 Calibration

Accuracy is one of the main metrics used in classification algorithms in data mining to favour an algorithm over others for certain data sets. In fact, most of classification problems such as credit card scoring, website classification, weather forecasting, etc., use accuracy or its complement one-error-rate as the main evaluation metric to distinguish among classification algorithms. Though, certain applications like cost-sensitive classification, Information Retrieval ranking in search engines, and text categorization for digital libraries, may require additional information beside classification accuracy such as class membership probabilities per test (Frunkranz, et al., 2008). So in calibrated AC approach, the derived rules per test data are used to describe the training data set and these rules are utilised to compute the class membership probabilities. When the rules are

accurate calibrated AC algorithms assumes that the estimated class membership probabilities are also accurate and can be generalised.

There are many classic rule based and non-rule based approaches in classification that have employed calibration. Some of which are SVM (Cortes and Vapnik, 1995) decision trees (Quinlan, 1998), and Statistical and probabilistic (Duda and Hart, 1973). In AC, one calibrated approach has been used AC, i.e. (Veloso, et al. 2011; Frunkranz, et al., 2008). We believe that calibration is an important issue that should be studied extensively in AC simply since initial results revealed good predictive performance if compared to other current algorithms. Furthermore, for multiple label classification including the class membership probabilities are much more useful than single label classification because of two reasons. Firstly, in multi-label classification, the input data instance may belong to several classes and therefore we can assign weights or class memberships in particular when classes overlap in the training data. Thus, the decision maker can distinguish easily to which the input data belongs to or can merge multiple classes together to come up with new class label. Secondly, some of the rules in the classifier will be connected to set of classes and therefore calibration can assist in prioritising these classes (Ranking).

7.3.4 Non Confidence based Learning

The key element, which controls the number of rules produced in AC is the support threshold. If the support is set to a large value, normally the number of extracted rules is very limited, and many rules with high confidence will be missed. This may lead to discarding important knowledge that could be useful in the classification step. To overcome this problem, one has to set the support threshold to a very small value. However, this usually involves the generation of massive number of classification rules, where many of which are useless since they hold low support and confidence values. This large number of rules may cause severe problems such as overfitting.

(Xu, et al., 2004) argued that the rule confidence which is the main criteria for selecting the classifier could be misleading in some cases especially since the rule with the largest confidence is chosen to predict the test case in the test data set. So, instead of computing the confidence from the training data set as most AC methods, the test data should be considered in favouring rules during the prediction phase. Therefore, the authors proposed a measure of rule goodness called “predictive confidence” which is based on statistical

information in the test data set (the frequencies of the test cases applicable to a rule). The new predictive confidence based AC approach is called AC-S. This approach is required to calculate the rule (R) "confidence decrease" = R(Conf(Training)) – R(Conf(Testing))" in order to estimate the predictive confidence for each rule before predicting test cases.

The AC-S algorithm depends on several parameters that must be known at the time of prediction and for each test case before the algorithm chooses the most applicable rule to the test case. Precisely, the support and confidence for each candidate rule must be computed and from both the training and testing data sets so that AC-S can be able to estimate the predictive accuracy for each rule. This indeed is time consuming and can be a burden in circumstances where the training data set is highly correlated. Further, it is impractical to estimate the support and confidence for each rule in the testing data set in advance since we don't know which rule will be used for prediction. Yet, we can utilise the test data during the prediction step to narrow down candidate rules. This can be seen a new research path for enhancing the current "predictive confidence" approach. A comparison between AC-S and other known AC algorithms such as CBA, CBA (2) and CMAR was conducted against some UCI data sets. The results of the accuracy showed that AC-S is competitive to CBA, though CBA (2) and CMAR algorithms derived higher quality classifiers than AC-S.

7.3.5 The Lift Rule Measure

There are few interesting measures besides confidence and support that can be used to evaluate the importance of the rules in both association rule discovery and AC. One of these measures is named the lift. Using the lift measure, one can interpret the significance of a rule since this measure is related to the rule rather than the itemset or the ruleitem. Let's define the lift in association rule discovery for a rule $R_1: x \rightarrow y$. The lift of R_1 corresponds to the ratio of R_1's confidence and the expected confidence of R_1. R_1's expected confidence can be defined as the product of the R_1's support and R_1's body frequency divided by the support R_1's body as shown in Equation (7.3). The general format of the lift measure is shown in Equation (7.2) below.

$$\text{Lift of a rule} = \frac{Confidence}{\exp ectedConfi dence} \qquad (7.2)$$

$$\text{Lift } (R_1) = \frac{P(X,Y)}{P(X)..P(Y)} = \frac{Confidence\,(x \rightarrow y)}{P(Y)} \qquad (7.3)$$

One of the shortcoming of the lift measure is that you cannot define a minimum value or a threshold for it as we normally do for the minimum confidence and minimum support in association rule and AC. It also worth to mention that lift has not been used in AC yet but possibly can be considered as an interesting method that discriminate among rules while constructing the classifier.

7.3.6 FP-Tree Data Representation

(Han, et al., 2000) presented an association rule discovery method called Frequent Pattern Growth (FP-Growth) that converts the transactional database into a condensed frequent pattern tree (FP-tree) in which each transaction corresponds to one path in the tree containing the frequent items in that transaction. The new representation of the input database (FP-tree) can be seen practical since frequent itemsets in each transaction are known by the tree, and the FP-tree is usually smaller in size than the complete input database because of the items sharing among frequent itemsets. Once the algorithm constructs the FP-tree, a pattern growth method kicks in to produce the rules from the FP-tree. For each frequent pattern *X*, the method uses links in the tree to derive other available patterns co-occurring with *X*, and then the FP-Growth algorithm concatenates *X* with the other patterns extracted from the FP-tree.

Figure 7.1 below contains a detailed example on FP-tree that shows its significance in mining association rules where normally the size of search space is significantly minimised if compared with the traditional Apriori based methods. This is one of the definite advantages of using FP-tree in AC mining since normally the size of the candidate ruleitems are massive particularly when the input data has many attributes or it is highly correlated. Therefore, utilising FP-Tree as a data representation format in AC will be a potential research direction that might results in a more efficient mining process because of the compact data structure this method offers.

.

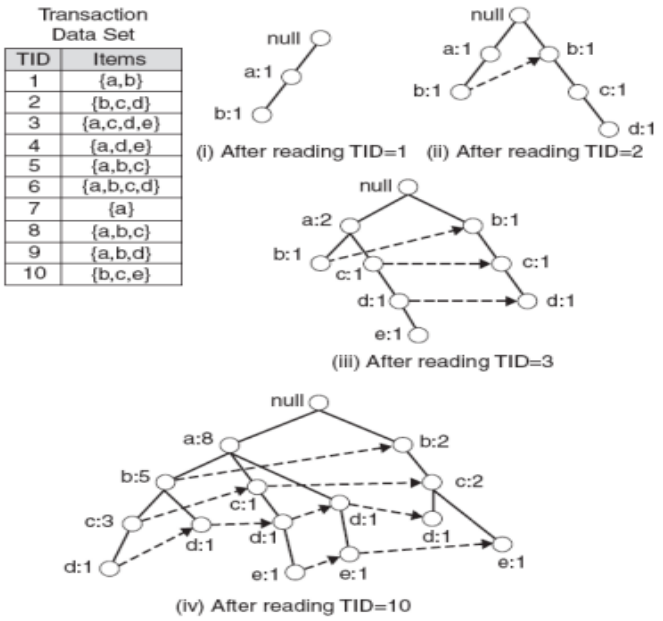

Fig.7.1 FP-Tree example from (Han, et al., 2000)

7.3.7 Group of Rules Prediction

The main step performed by the AC algorithm is to allocate the right class from one of the classifier rules to the test data as accurately as possible. This is named the classification step. Different methods for class allocation in AC exist some of which employs the first rule in the classifier like CBA and MCAR, and others employs more than one rule like CMAR as discussed early in Chapter 2. We highlight in this section class allocation methods that use group of rules prediction as a possible research path to improve the overall performance of AC classifiers.

Different criteria can be utilised to discriminate among relevant rules applicable to the test cases during the classification steps. For instance, some scholars initiated the use of

group of rules based on the confidence value in which they have marked all rules matching the test case at the first instance. Then, these rules have been clustered based on the class labels and the class belongs to the group having the highest average confidence is allocated to the test case. Another good initial research work on the group of rules prediction have utilised both the support and confidence combinations where the class linked with the group of rules that have the largest (Support and Confidence) gain gets assigned to the test case.

We believe that the group of rules prediction in the classification phase is a step forward toward enhancing the overall predictive performance in AC for two main reasons. Firstly, it is a fairer decision since the single rule biased decision is eliminated when using more than one applicable rule. Secondly, larger rules set is used which legitimate the decision and reduces the chance of misclassification.

7.3.8 Minority vs Majority Class in Rule Ranking

In AC, rule ranking plays a vital role in determining the rules that will end up in the classifier which in turns is employed to assign the appropriate class to test cases. Thus, ranking is a fundamental step in AC that often influences the classifier performance at the end. Another major reason why ranking rules is essential is the fact that most AC algorithms derive large set of rules. These rules often share common confidence, support, and length and therefore selecting the precedence of the rules mechanism becomes a hard task.

There have been good research attempts in the literature of AC to deal with the rules precedence problem by imposing more tie breaking criteria to distinguish among rules in the ranking process and thus minimise any random decision. Two main criteria that has been added to deal with reducing rule random decisions are the minority and the majority class. The latter criterion favours rules that are connected with the highest frequent class in the training data when these rules turn to have identical confidence, support and length. On the other hand, the former criterion prefers rules connected with the lowest frequency class in the training data.

The minority and majority class solutions in rule ranking have their pros and cons. For instance, approaches that prefer the majority class as a tie breaking criterion claim that the

class with the highest frequency most likely to be the more accurate class. Though, normally, this class is derived as a default rule from the unclassified cases in the training data set and therefore it will never get ignored. The other side which favours the minority class claims that the least frequent class in the training data set is not well represented in the context of rules in the classifier and should be given a higher priority in the ranking process. Overall, there have been few experimental studies that reveal slight improvement in the classifier performance when using the minority class, i.e. (Abdelhamid, et al., 2012).

Bibliography

[1] Aaron G., Manning R. (2012) APWG Phishing Reports. Retieved March 12, 2013 from Http://www.antiphishing.org/resources/apwg-reports/.

[2] Abdelhamid N., Ayesh A., Thabtah F. (2013a) Associative Classification Mining for Website Phishing Classification. *Proceedings of the ICAI '2013*,pp. *687-69.* USA.

[3] Abdelhamid N., Ayesh A., Thabtah F. (2012a) An Experimental Study of Three Different Rule Ranking Formulas in Associative Classification Mining. *Proceedings of* the *7th International Conference for Internet Technology and Secured Transactions (ICITST-2012).*

[4] Abdelhamid N., Ayesh A., Thabtah F., Ahmadi S., Hadi W. (2012b) MAC: A multiclass associative classification algorithm. *Journal of Information and Knowledge Management (JIKM). 11 (2), 1250011-1 - 1250011-10. Worldscinet.*

[5] Abdelhamid N., Ayesh A., Thabtah F., Hadi W., Ahmadi S., (2011) A new multiclass associative classification data mining algorithm. *Proceedings of the 2011 Conference on Innovations in Computing and Engineering Machinery (CICEM 2011).*

[6] Abumansour H., Hadi W., mccluskeyL., Thabtah F. (2010) Associative Text Categorisation Rules Pruning Method. *Proceedings of the Linguistic and Cognitive Approaches to Dialog Agents Symposium (lacatoda-10), RafalRzepka (Ed.), at the AISB 2010 convention,* 39-44

[7] Aburrous M., Hossain M.A., Dahal K., Thabtah F. (2010a) Predicting Phishing Websites using Classification Mining Techniques. *In Seventh International Conference on Information Technology;* 2010, *IEEE*, 176-181.

[8] Aburrous M., Hossain MA., Dahal K., Thabtah F. (2010b)Intelligent phishing detection system for e-banking using fuzzy data mining. *Expert Systems with Applications: An International Journal,*7913-7921.

[9] Afroz S., Greenstadt R. (2011) PhishZoo: Detecting Phishing Websites by Looking at Them. In Proceedings of the 2011 IEEE Fifth International Conference on Semantic Computing (ICSC '11). *IEEE Computer Society*, Washington, DC, USA, 368-375

[10] Agrawal R., Srikant R. (1994) Fast algorithms for mining association rule. *Proceedings of the 20th International Conference on Very Large Data Bases-VLDP,*487-499.

[11] Alexa the Web Information Company (2011) Alexa the Web Information Company. Retieved February 22, 2013 from Http://www.alexa.com/.

[12] Al-Maqaleh B. (2013) Discovering Interesting Association Rules: A Multi-objective Genetic Algorithm Approach. *International Journal of Applied Information Systems* 5(3):47-52.

[13] Anon, Gartner, Inc. (2011) Retieved February 2, 2013 fromhttp://www.gartner.com/technology/home.jsp

[14] Antonie M., Zaïane O. (2004) An associative classifier based on positive and negative rules. *Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 64 - 69.

[15] Antonie M., Zaïane O. (2002) Text documents categorization by term association. *Proceedings of theIEEE International Conference on Data Mining*, 19-26.

[16] Arunasalam B., Chawla S. (2006) CCCS: a top-down associative classifier for imbalanced class distribution. KDD 2006: 517-522.

[17] Baralis E., Chiusano S., Graza P. (2008) A Lazy Approach to Associative classification. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING. Vo. 20, num 2. ISSN: 1041-4347.*

[18] Baralis E., Chiusano S., Graza P. (2004) On support thresholds in associative classification.*Proceedings of the ACM Symposium on Applied Computing*, 553-558.

[19] Baralis E., Torino P. (2002) A lazy approach to pruning classification rules. *Proceedings of the 2002 IEEE ICDM'02*,35.

[20] Bouker S., Saidi R., Ben Yahia S., Nguifo E. M. (2012) Ranking and Selecting Association Rules Based on Dominance. *ICTAI 2012*, 658-665.

[21] Cendrowska, J. (1987) PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, Vol.27, No.4, 349-370.

[22] Cerf L., Gay D., Selmaoui N., Boulicaut F. (2008) A Parameter-Free Associative Classification Method**.** *I.-Y. Song, J. Eder, and T.M. Nguyen (Eds.): Dawak, LNCS* 5182, 293–304.

[23] Chakhlevitch K., Cowling P. (2008) Hyperheuristics: Recent Developments (2008) *Adaptive and Multilevel Metaheuristics* 2008: 3-29.

[24] Chen C., Chiang D., C. Chen (2012) Improving the Performance of Association Classifiers by Class Prioritization. Journal of Computational Information Systems 8: 4,

1697- 1712.

[25] Chen, J. Guo, C. (2006) Online Detection and Prevention of Phishing Attacks. IEEE Communications and Networking, China Com '06, 1-7.

[26] Chen, J., Yin, J., Huang, J. (2005) Mining Correlated Rules for Associative Classification. *In Proceedings of ADMA*. 2005, 130-140.

[27] Chien Y., Chen Y. (2010) Mining associative classification rules with stock trading data – A GA-based method. *Knowledge-Based Systems*23, 605–614.

[28] Clare A., King R. (2001) Knowledge discovery in multi-label phenotype data. In L. De Raedt and A. Siebes, editors, *Proceedings of the PKDD '01*, volume 2168 of Lecture Notes in Artificial Intelligence, 42-53.

[29] Cohen W. (1995) Fast effective rule induction. *Proceedings of the 12th International Conference on Machine Learning*, 115-123.

[30] Cortes C., Vapnik V. (1995). Support-Vector Networks. *Machine Learning, 20* (3), 273 - 297.

[31] Costa G., Ortale R., Ritacco E. (2013) X-Class: Associative Classification of XML Documents by Structure. *ACM Trans. Information Syst*ems 31(1): 3.

[32] Coulter M. (2012) Strategic Management in Action. Pearson Education, 2012.

[33] Cowling P., Chakhlevitch K. (2007) Using a Large Set of Low Level Heuristics in a Hyperheuristic Approach to Personnel Scheduling. *Evolutionary Scheduling*,543-576.

[34] Dede (2011) http://blog.sucuri.net/tag/blacklisted (accessed June 25, 2013)

[35] Dhamija R., Tygar JD., Hearst M. (2006) Why Phishing Works. *In Proceedings of the SIGCHI conference on Human Factors in computing systems;* Cosmopolitan Montréal, Canada: ACM., 581-590.

[36] Dhamija R, Tygar J. (2005) The battle against phishing Dynamic Security Skins. *In Proceedings of the 1st Symposium On Usable Privacy and Security*; New York, NY, USA: ACM Press., 77-85.

[37] Dhok J., Varma V, (2010) Using Pattern Classification for Task Assignment in mapreduce. Proceedings of the 10th IEEE/ACM International Conference ccgrid.

[38] Do, T.D., Hui, S.C., Fong, A.C.M., Fong, B. (2009) Associative classification with artificial immune system. *IEEE Transactions on Evolutionary Computation* 13, 217–228

[39] Dong G., Zhang X., Wong L., Li J., (1999) CAEP: Classification by aggregating

emerging patterns. *Proceedings of the Second Imitational Conference on Discovery Science*, 30-42. Tokyo, Japan.

[40] Duda R., Hart P. (1973) Pattern classification and scene analysis. John Wiley & Son.

[41] Elsayed S., Rajasekaran S., Ammar R. (2012) AC-CS: An Immune-Inspired Associative Classification Algorithm.*ICARIS,* 139-151.

[42] Furnkranz J., Ullermeier E. H, Menc´ıaE. L., Brinker K. (2008) Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153.

[43] Frank, E., Witten, I. (1998) Generating accurate rule sets without global optimisation. *Proceedings of the Fifteenth International Conference on Machine Learning,* (p. . 144–151). Madison, Wisconsin.

[44] Gehrke, J., Ramakrishnan, R., Ganti, V. (1998) *Rainforest:* A Framework for fast decision tree construction of large datasets. *Proceedings of VLDB*, 416-427. .

[45] Greensmith J., Aickelin U., Cayzer S (2005) Introducing Dendritic Cells as a Novel Immune-Inspired Algorithm for Anomaly Detection. *In: Jacob, C., Pilat, M.L., Bentley, P.J., Timmis, J.I. (eds.) ICARIS 2005. LNCS,* vol. 3627, 153–167. Springer, Heidelberg.

[46] Guang X, Jason O , Carolyn P R, Lorrie C. (2011) CANTINA+: A Feature-rich Machine Learning Framework for Detecting Phishing Web Sites. *ACM Transactions on Information and System Security*, 1-28.

[47] Hammoud S. (2010) A MapreduceClassifier based on association rule Data Mining. PhD thesis, Brunel University.

[48] Han J., Pei J., Yin Y. (2000) Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 1-12.

[49] Holte R.C. (1993) Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. Machine Learning, 11, pp 63-90.

[50] Hooshsadat M., Zaiane O. (2012) An Associative Classifiers for Uncertain Datasets. Proceedings of the 16th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKFF 2012), 342-353, Malaysia.

[51] Horng SJ., Fan P., Khan MK, Run RS, Lai JL, Chen RJ. (2011) An efficient phishing webpage detector. Expert Systems with Applications: *An International Journal*, 38(10): 12018-12027.

[52] Jabbar M. A., Deekshatulu B. L. Chandra P. (2013) Knowledge Discovery Using Associative Classification for Heart Disease Prediction. *Advances in Intelligent Systems and Computing Volume* 182, 29-39.

[53] Jabez C. (2011) A statistical approach for associative classification. *European Journal of Scientific Research* Vol. 58 (No. 2) 140-147.

[54] James , L. (2005) Phishing Exposed. s.l.:Syngress Publishing.

[55] Ganesh-Kumar V., Muneeswaran K. (2013) intelligent apriori algorithm for complex activity mining in supermarket applications**.** *Journal of Computer Science,* Volume 9, Issue 4, 433-438.

[56] Kaspersky Lab (2013)Retrieved March 15, 2013 from http://www.kaspersky.com/about/news/spam/2013/.

[57] Kirda E. Kruegel C. (2005) Protecting users against phishing attacks with antiphish. *In Proceedings of the 29th Annual International Computer Software and Applications Conference* (COMPSAC), 517–524.

[58] Kundu G., Islam M., Munir S., Bari M. (2008). ACN: An Associative Classifier with Negative Rules, 11th *IEEE International Conference on Computational Science and Engineering.*369-375.

[59] Kunz M., Wilson P. (2004) Computer Crime and Computer Fraud. Technical Report. Submitted to the Montgomery County Criminal Justice Coordinating Commission.

[60] Lan Y., Janssens D., Chen G., Wets G. (2006) Improving associative classification by incorporating novel interestingness measures.*Expert System Applications*, 184-192.

[61] Li X., Qin D., Yu C. (2008) ACCF: Associative Classification Based on Closed Frequent Itemsets. *Proceedings of the Fifth International Conference on Fuzzy Systems and Knowledge Discovery– FSKD*, 380-384.

[62] Li W., Han J., Pei J. (2001) CMAR: Accurate and efficient classification based on multiple-class association rule. *Proceedings of the IEEE International Conference on Data Mining –ICDM*, 369-376.

[63] Liu B., Ma Y., Wong C-K. (2001) Classification using association rules: weakness and enhancements. *In Vipin Kumar, et al, (eds) Data mining for scientific applications*.

[64] Liu B., Hsu W., Ma Y. (1998) Integrating classification and association rule mining. *Proceedings of the Knowledge Discovery and Data Mining Conference- KDD,* 80-86. New York.

[65] Liu J., Ye Y. (2001) Introduction to E-Commerce Agents: Marketplace Solutions, Security Issues, and Supply and Demand. *In E-Commerce Agents, Marketplace Solutions, Security Issues, and Supply and Demand*, 1-6.

[66] Liu W., Huang G., Xiaoyue L., Min Z., Deng X. (2005) Detection of Phishing Webpages based on Visual Similarity. *In Proceeding '05 Special interest tracks and posters of the 14th international conference on World Wide Web , ACM.*, 1060 - 1061.

[67] Merz, C., Murphy, P. (1996) UCI repository of machine learning databases. Irvine, CA, University of California, Department of Information and Computer Science.

[68] Mei Q., Xin D., Cheng H., Han J., Zhai, C. X. (2006) Generating semantic annotations for frequent patterns with context analysis, *In Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining,* 337–346.

[69] Millersmiles (2011) millersmiles. Retieved March 2, 2013 from Http://www.millersmiles.co.uk/.

[70] Miyamoto D. ,Hazeyama H., Kadobayashi Y. (2008) An Evaluation of Machine Learning-based Methods for Detection of Phishing Sites. *Australian Journal of Intelligent Information Processing Systems,* 54-63.

[71] Mohammad R., Thabtah F., McCluskey L. (2012) An Assessment of Features Related to Phishing Websites using an Automated Technique. *In The 7th International Conference for Internet Technology and Secured Transactions (ICITST-2012);*

[72] Niu Q., Xia S. Zhang L. (2009). Association Classification Based on Compactness of Rules. *Proceedings of the Second International Workshop on Knowledge Discovery and Data Mining - WKDD,* 245-247.

[73] Pal P.R., Jain R.C (2010) Combinatorial Approach of Associative Classification. *International Journal of Advanced Networking & Applications,* Vol. 2 Issue 1, 470.

[74] Pan Y., Ding X. (2006) Anomaly Based Web Phishing Page Detection. *Proceedings of the 22nd Annual Computer Security Applications Conference*.; *IEEE,* 381-392.

[75] Phishtank (2006) phishtank. Retrievd January 19 2013 from Http://www.phishtank.com/.

[76] Quinlan J. (1998) Data mining tools See5 and C5.0. Technical Report, rulequestResearch.

[77] Quinlan J. (1993) C4.5: Programs for machine learning. San Mateo, CA: Morgan Kaufmann.

[78] Quinlan J., Cameron-Jones R. (1993) FOIL: A midterm report. *Proceedings of the European Conference on Machine Learning*, (pp. 3-20), Vienna, Austria.

[79] Quinlan, J. (1979) Discovering rules from large collections of examples: a case study. In D. Michie, editor, *Expert Systems in the Micro-electronic Age*, (pp.168-201). Edinburgh, 1979.

[80] Qin, XJ, Zhang, Y, Li, X Wang, Y (2010). Associative classifier for uncertain data. *11th International Conference on Web-Age Information Management (WAIM 2010),*692-703.

[81] Ramaswami M., Bhaskaran R. (2009) A study on feature selection techniques in educational data mining. *International Journal Advanced Computer Science and Application* 2(1): 7–11.

[82] Rameshkumar K., Sambath M., Ravi S. (2013) Relevant association rule mining from medical dataset using new irrelevant rule elimination technique. *Proceedings of ICICES* , 300-304.

[83] Rak R., Kurgan L., Reformat M. (2005) Multi-label associative classification of medical documents from medline. *In Proceedings of the 4th International Conference on Machine Learning and Applications*, 177–186.

[84] Read J., Bifet A., Holmes G., Pfahringer B. (2011) Streaming multi-label classification. *JMLR Workshop and Conference Proceedings (Second Workshop on Applications of Pattern Analysis*) 17, 19.

[85] Sadeh N., Tomasic A., Fette I. (2007) Learning to detect phishing emails. *Proceedings of the 16th international conference on World Wide Web.*649-656.

[86] Sanglerdsinlapachai N., Rungsawang A. (2010) Using Domain Top-page Similarity Feature in Machine Learning-based Web. *In Third International Conference on Knowledge Discovery and Data Mining,* 187-190.

[87] Schapire R.E. Singer Y. (2000) "Boostexter: a boosting-based system for text categorization," *Machine Learning*, vol. 39, no. 2/3, 135–168.

[88] Sheng S., Wardman B., Warner G., Cranor L. F., Hong J., Zhang C. (2009) An empirical analysis of phishing blacklists, *CEAS.*

[89] Seogod (2011) Black Hat SEO.

[90] Shekhawat P.B., Dhande S.S (2011) A classification technique using associative classification. *International journal of computer application*(0975-8887) vol. 20-No.5,20-28.

[91] Song M. (2009) Handbook of research on text and web mining technologies, information science reference, *IGI global*.

[92] Su Z., Song W., Cao D., Li J. (2008) Discovering Informative Association Rules for Associative Classification. Proceedings of the Knowledge Acquisition and Modeling Workshop - KAM Workshop, 1060-1063.

[93] Taiwiah C. A., V. Sheng (2013) A study on Multi-label Classification. Advances in Data Mining. Applications and Theoretical Aspects. Lecture Notes in Computer Science, Volume 7987, 137-150.

[94] Tang Z. Liao Q. (2007). A New Class Based Associative Classification Algorithm. *IMECS* ,685-689.

[95] Thabtah F., Hammoud S (2013) MR-ARM: A MapReduce Association Rule Mining. *Journal of Parallel Processing Letter*, 23, 1350012. World Scientific.

[96] Thabtah F., Hadi W., Abdelhamid N., (2011) Prediction Phase in Associative Classification. *Journal of Knowledge Engineering and Software Engineering*. Volume: 21, Issue: 6, 855-876. Worldscinet.

[97] Thabtah F., Mahmood Q., mccluskeyL., Abdel-Jaber H (2010). A new Classification based on Association Algorithm. *Journal of Information and Knowledge Management*, Vol 9, No. 1, 55-64. World Scientific.

[98] Thabtah F., Eljinini M., Zamzeer M., Hadi W. (2009) Naïve Bayesian based on Chi Square to Categorize Arabic Data. *In proceedings of The 11th International Business Information Management Association Conference (IBIMA) Conference On Innovation and Knowledge Management in Twin Track Economies*, 930-935.

[99] Thabtah, F. Cowling, P. (2007) A greedy classification algorithm based on association rule. Applied Soft Computing, 7 (3), 1102-1111.

[100] Thabtah F. (2007): Review on Associative Classification Mining. *Journal of Knowledge Engineering Review*, Vol.22:1, 37-65. Cambridge Press.

[101] Thabtah F., Cowling P., Peng Y. (2005) MCAR: Multi-class classification based on association rule approach. *Proceedings of the 3rd IEEE International Conference on Computer Systems and Applications* , 1-7.

[102] Thabtah, F., Cowling, P., Peng, Y. (2004) MMAC: A new multi-class, multi-

label associative classification approach. *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM '04*), 217-224.

[103] The Government of Australia (2011) Hackers, Fraudsters and Botnets: Tackling the Problem of Cyber Crime. Report on Inquiry into Cyber Crime, 2011.

[104] Tsoumakas G., Katakis I., Vlahavas I. (2010) Mining multi-label data. In O. Maimon and L. Rokach,editors, *Data Mining and Knowledge Discovery Handbook.* Springer, Berlin.

[105] Vaithiyanathan, V.,Rajeswari, K., Phalnikar, R., Tonge, S. (2012) Improved apriori algorithm based on selection criterion. *Proceedings of the Computational Intelligence & Computing Research (ICCIC '2012),* 1-4.

[106] Veloso A., Meira W., Zaki M., Goncalves M. Mossri H. (2011). Calibrated Lazy Associative Classification. Information Sciences: an International Journal, Volume 13 (181), 2656-2670.

[107] Veloso A., Meira W., Gonçalves M., Zaki M (2007) Multi-label Lazy Associative Classification. Proceedings of the Principles of Data Mining and Knowledge Discovery, PKDD, 605-612.

[108] Wang X., Yue K., Niu W., ShiZ. (2011) An approach for adaptive associative classification. *Expert Systems with Applications: An International Journal*, Volume 38 Issue 9, 11873-11883.

[109] WEKA (2011): Data Mining Software in Java: Retrieved December 15, 2010 from http://www.cs.waikato.ac.nz/ml/weka.

[110] Whois (2011) Available from: http://who.is/

[111] Witten I., Frank E. (2002) Data mining: practical machine learning tools and techniques with Java implementations. San Francisco: Morgan Kaufmann.

[112] Wu C. H., Wang J.Y. , Chen C. J. (2012) Mining condensed rules for associative classification. *ICMLC 2012,* 1565-1570

[113] Wu G., Li H., Hu X., Bi Y., Zhang J. Wu X. (2009) mrec4.5: C4.5 Ensemble Classification with mapreduce. *Proceedings of thechinagrid Annual Conference*, 249-255

[114] Xu X., Han G., Min H. (2004) A novel algorithm for associative classification of images blocks. Proceedings of the fourth IEEE International Conference on Computer and Information Technology, 46-51

[115] Yang B. Yang, J. Sun, T. Wang, Chen Z. (2009) Effective multi-label active learning for text classification. *In Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 917–926.

[116] Ye Y., Jiang Q., Zhuang W. (2008) Associative classification and post-processing techniques used for malware detection. Proceedings of the 2nd International Conference on Anti-counterfeiting, Security and Identification, ASID, 276-279.

[117] Yin X., Han J. (2003) CPAR: Classification based on predictive association rule. *Proceedings of the SIAM International Conference on DataMining -SDM*, 369-376.

[118] Yoon Y., Lee G. (2008) Efficient implementation of associative classifiers for document classification, *InformationProcessing Management, An International Journal*, 43(2): 393-405.

[119] Yu K., Wu X., Ding W., Wang H. (2011) Causal Associative Classification, Proceedings of the 11th IEEE International Conference on Data Mining (ICDM '11) 914-923.

[120] Zaki, M., Gouda, K. (2003) Fast vertical mining using diffsets. *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 326 – 335.

[121] Zaki M., Hsiao CJ (2002) CHARM: an efficient algorithm for closed itemset mining. P*roceedings of the 2002SiamInternationalconference on data mining (SDM'02)*,457–473.

[122] Zaki, M., Parthasarathy, S., Ogihara, M., Li, W. (1997) New algorithms for fast discovery of association rules. *Proceedings of the3rd KDD Conference* ,283-286.

[123] Zhang Y., Hong J., Cranor L. (2007) CANTINA: A Content-Based Approach to Detect Phishing Web Sites. *In Proceedings of the 16th World Wide Web Conference*, 649-656

[124] Zhao Z., Ma H., He Q. (2009) Parallel K-Means clustering based on mapreduce. *Cloud Computing, Lecture Notes in Computer Science, Volume 5931. Springer-Verlag* Berlin, 674.

[125] Zhu Y., Luo W., Cheng J., Ou J. (2012) A Multi-label Classification Method Based on Associative Rules. *Journal of Computational Information Systems*, 791–799.

Appendix A

Sample Screen Shots of the Program and Weka Software

Sample of Screen Shots of the Program.

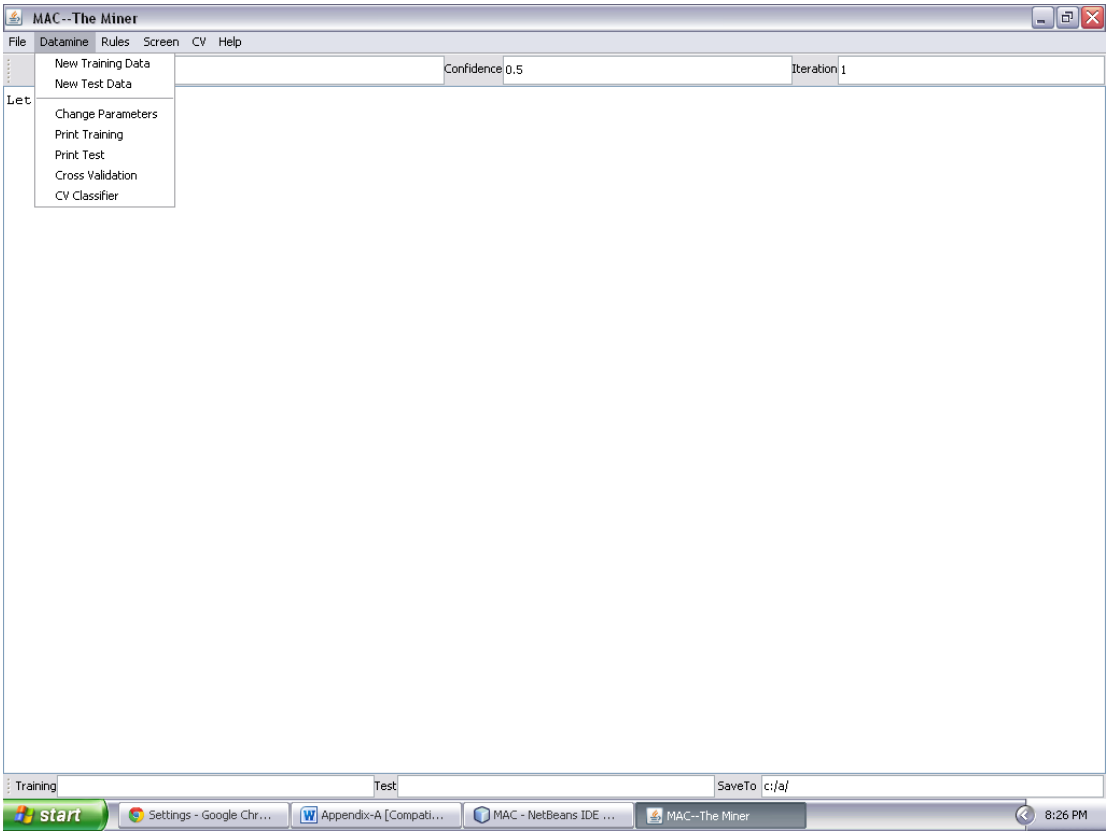

Fig. 1 General working Environment of MAC

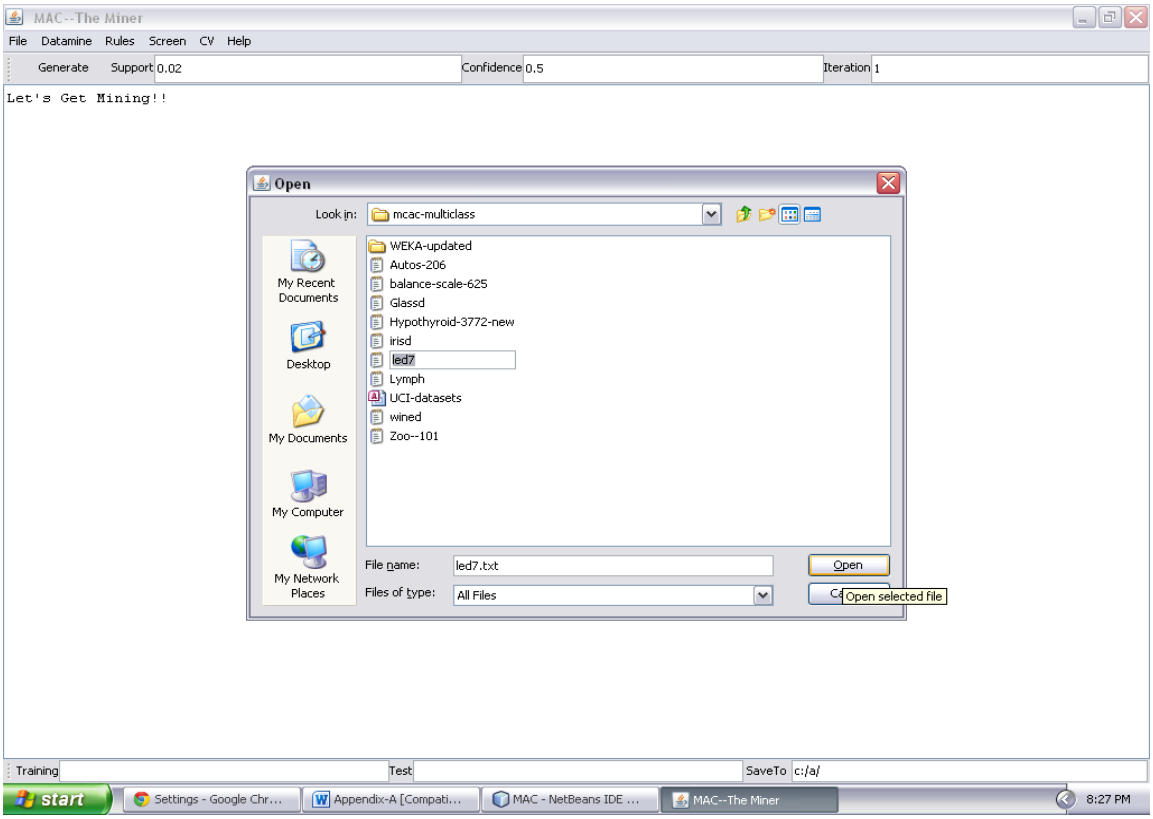

Fig. 2 Loading of a training data set (LED7)

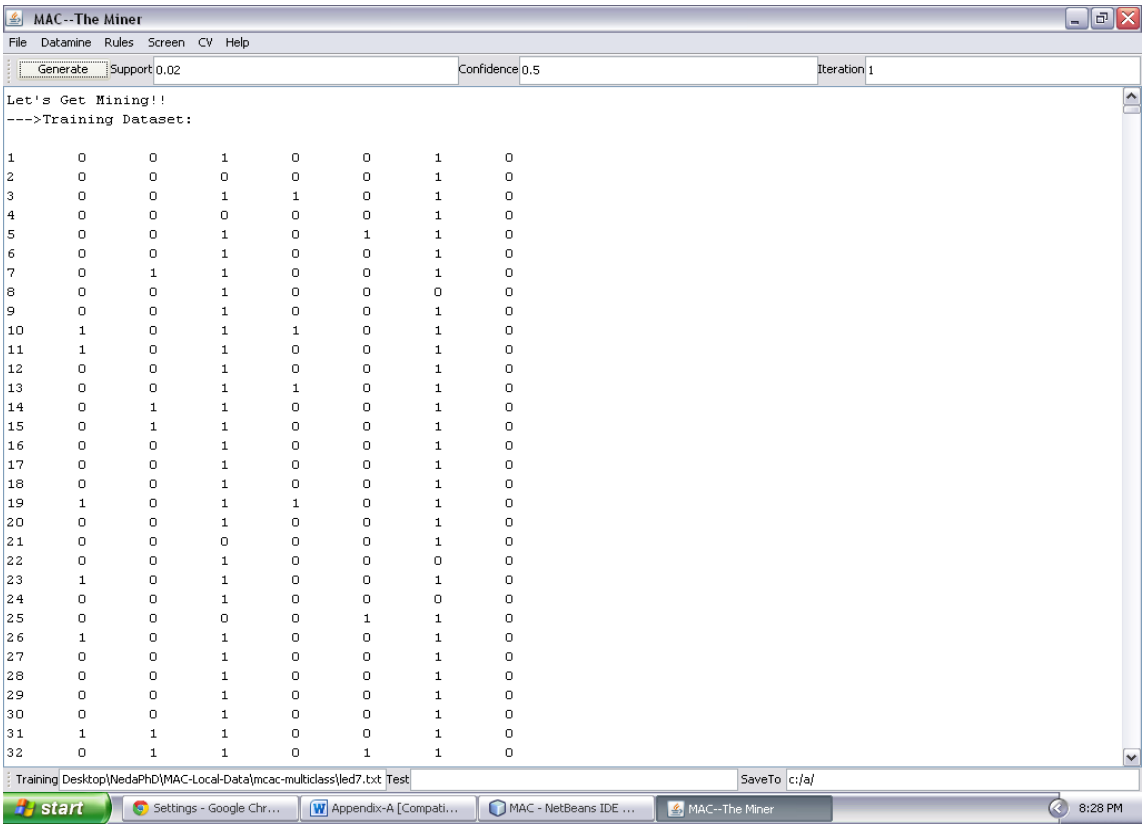

Fig. 3 Sample of the "LED" data sets after loading

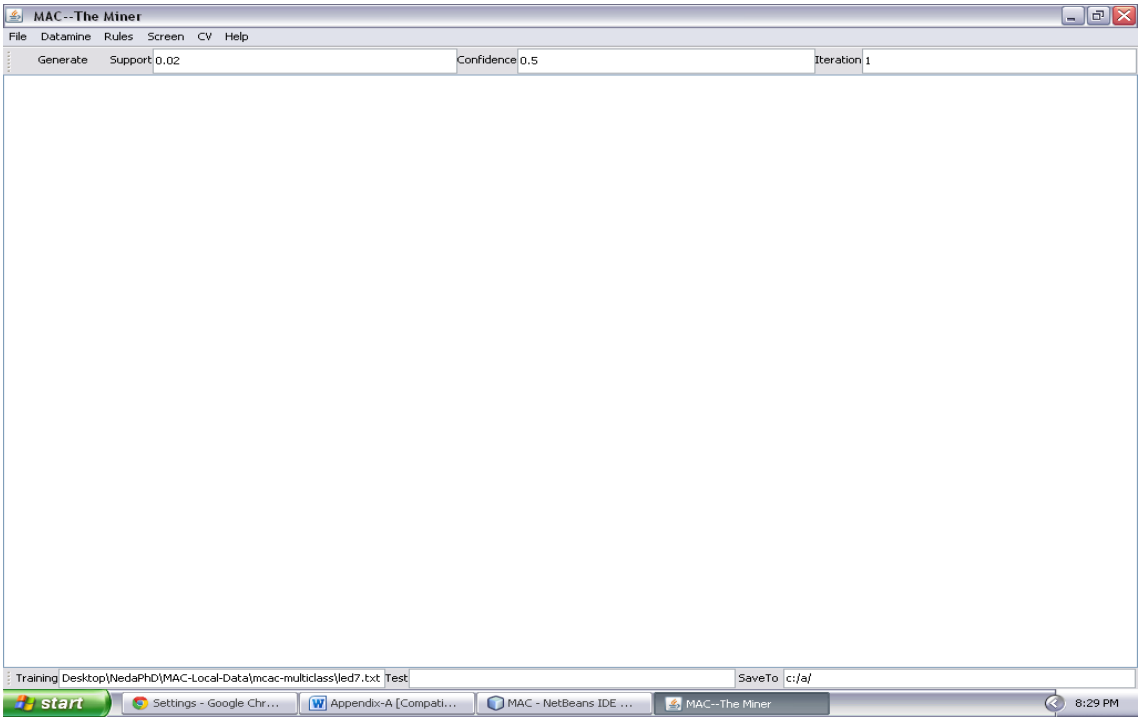

Fig. 4 Setting minimum support and minimum confidence to 2% and 50% respectively

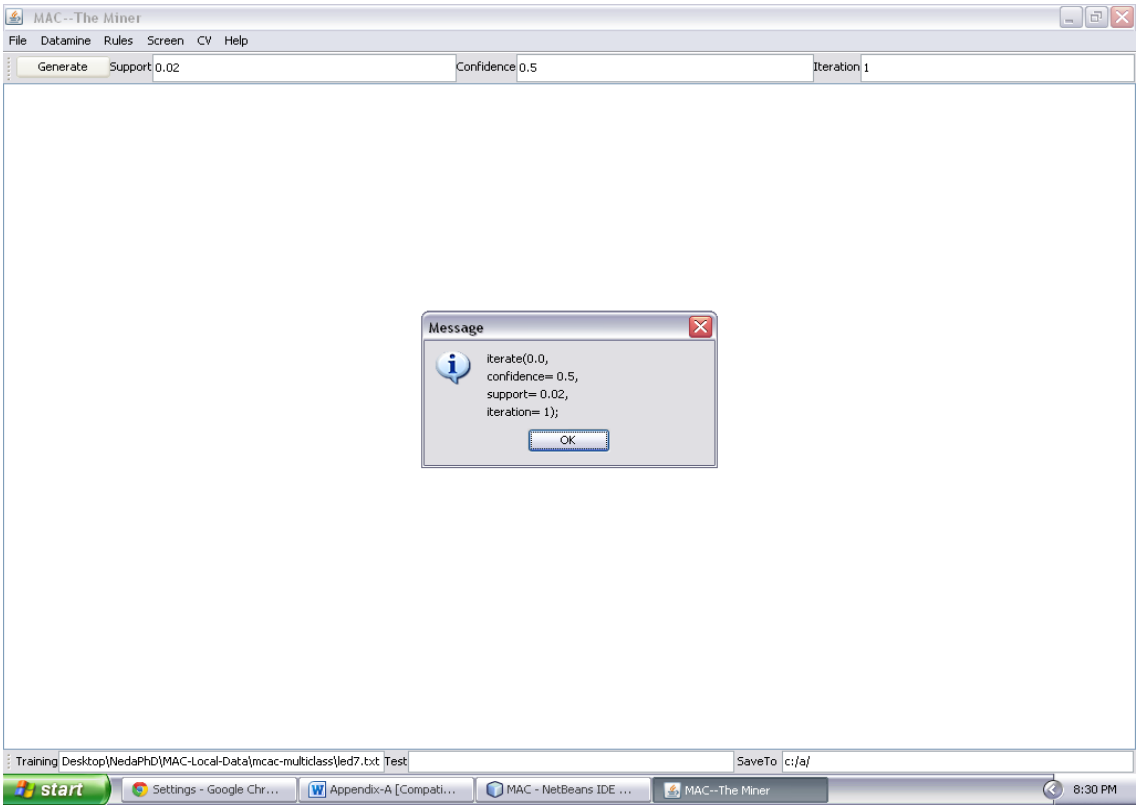

Fig. 5 Building the classifier

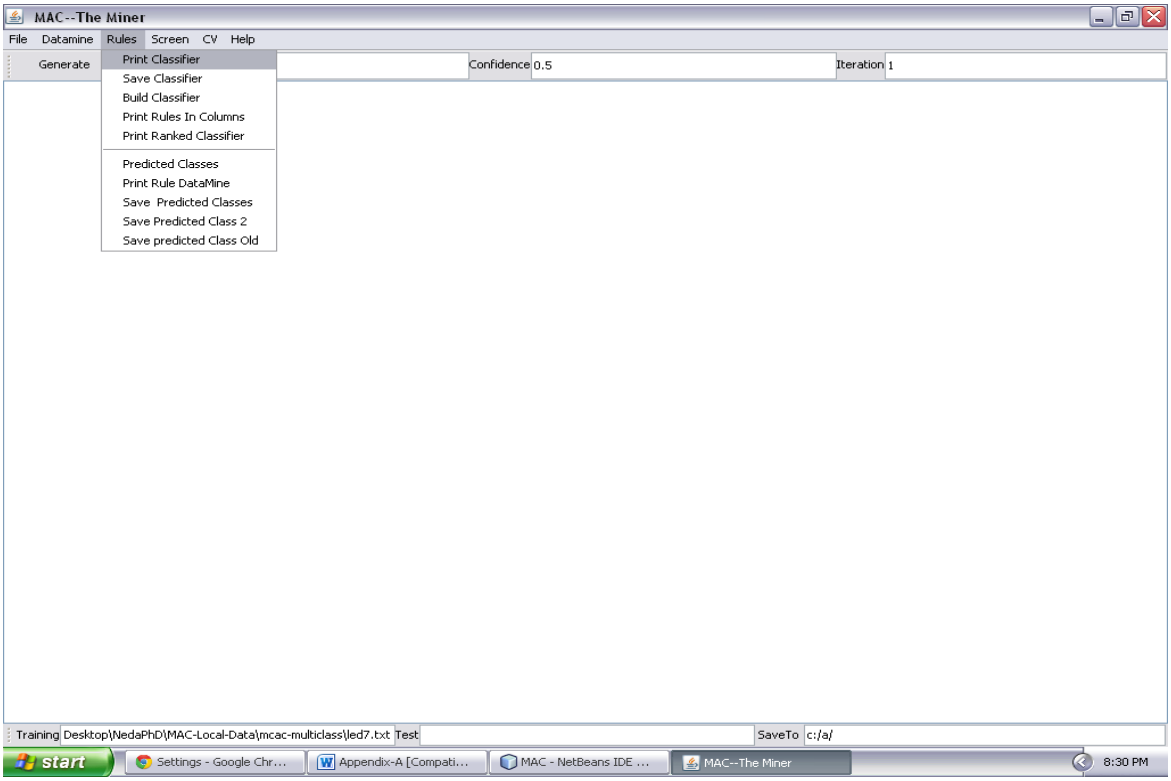

Fig. 6 Printing the classifier rules after building it

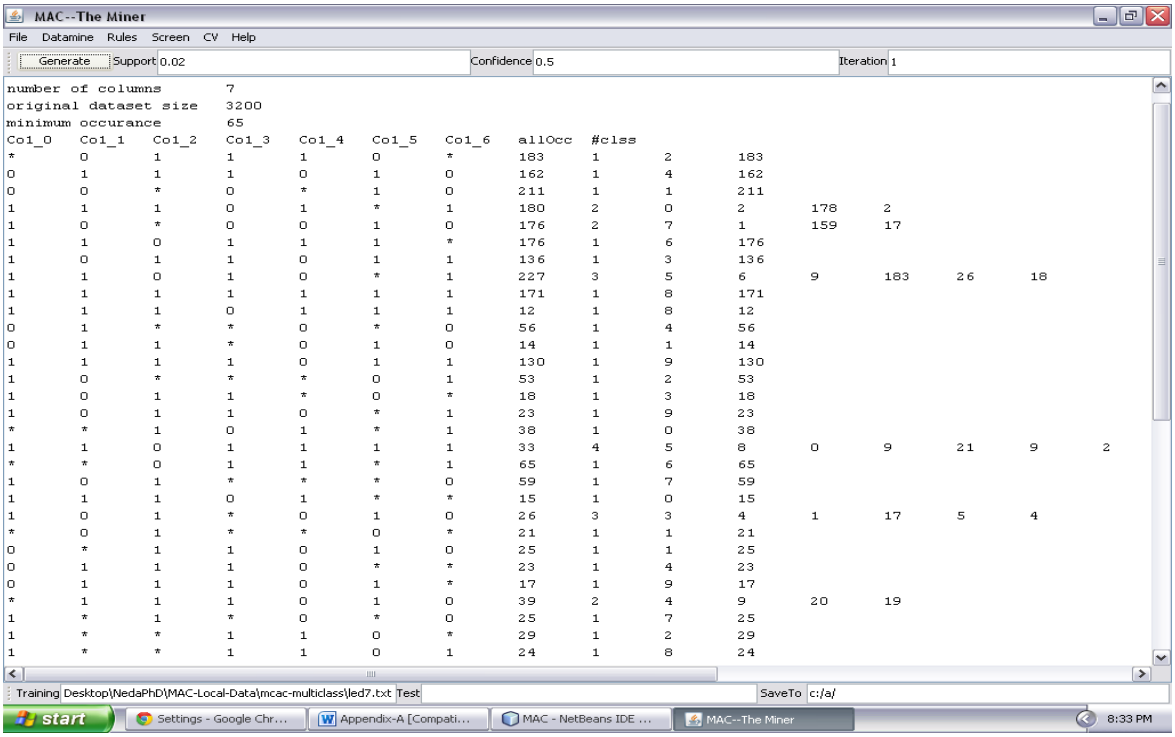

Fig. 7 Displaying the classifier rules after selecting the “Print Classifier” option

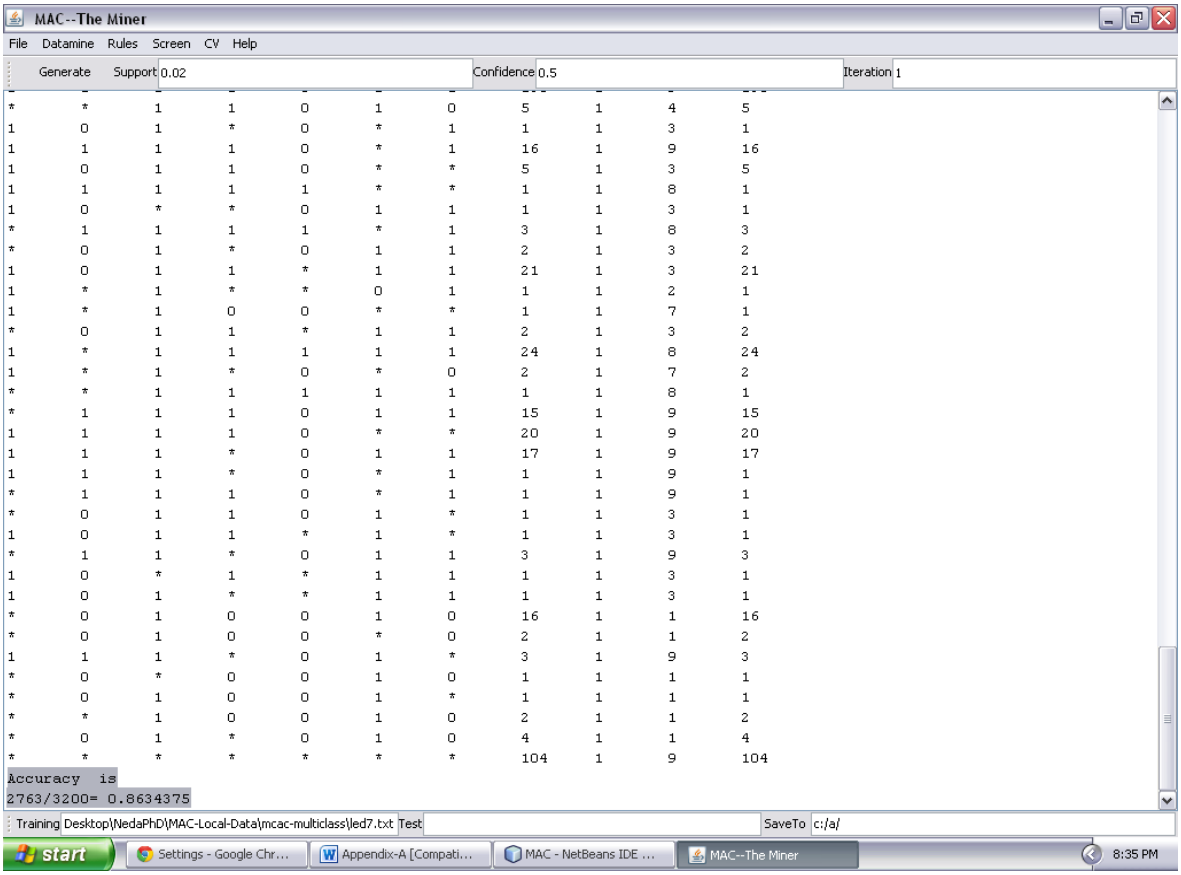

Fig. 8 The calculated accuracy in % for the classifier performance on the Training data.

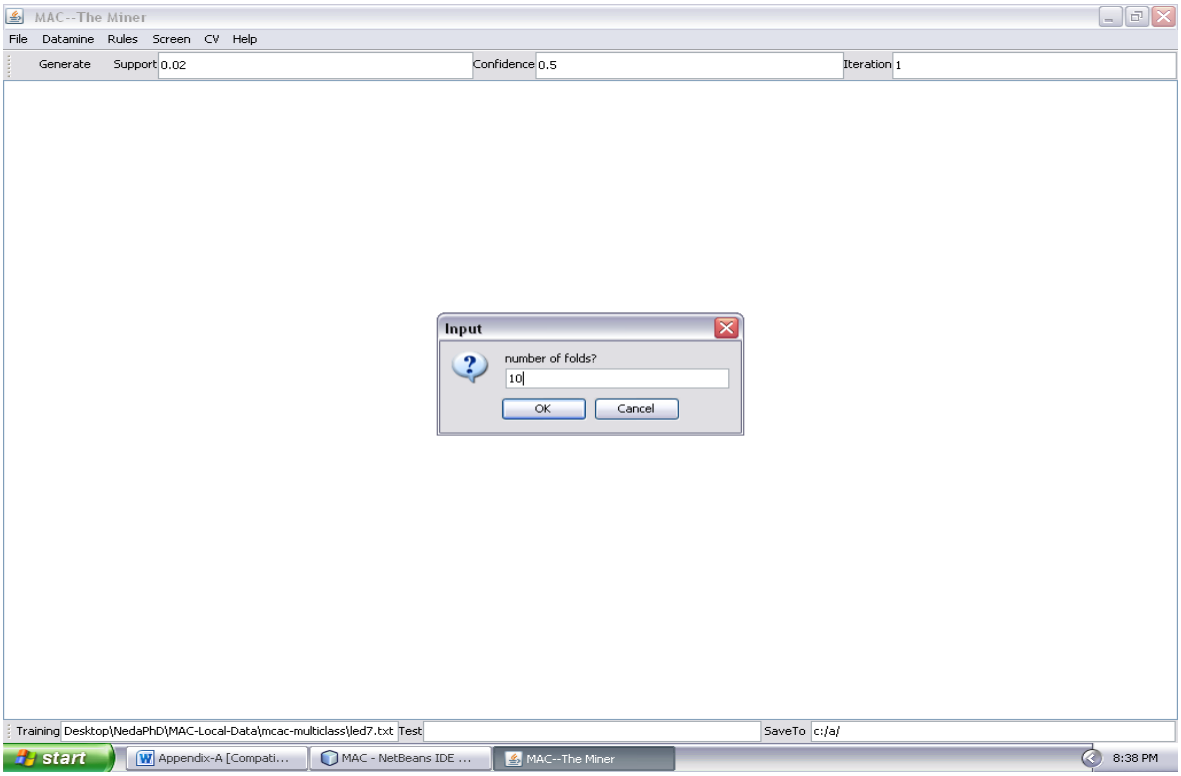

Fig. 9 Performing cross validation to produce classifier performance using evaluation measures

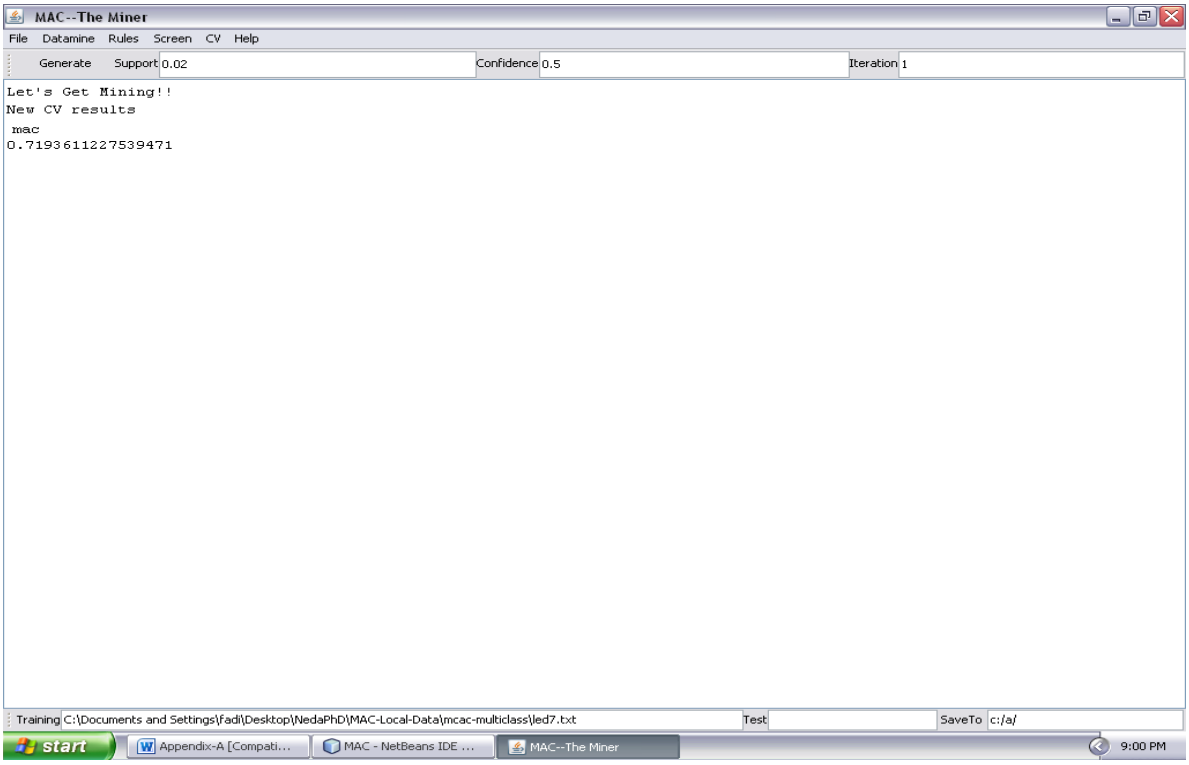

Fig. 10 Results on different evaluation measures

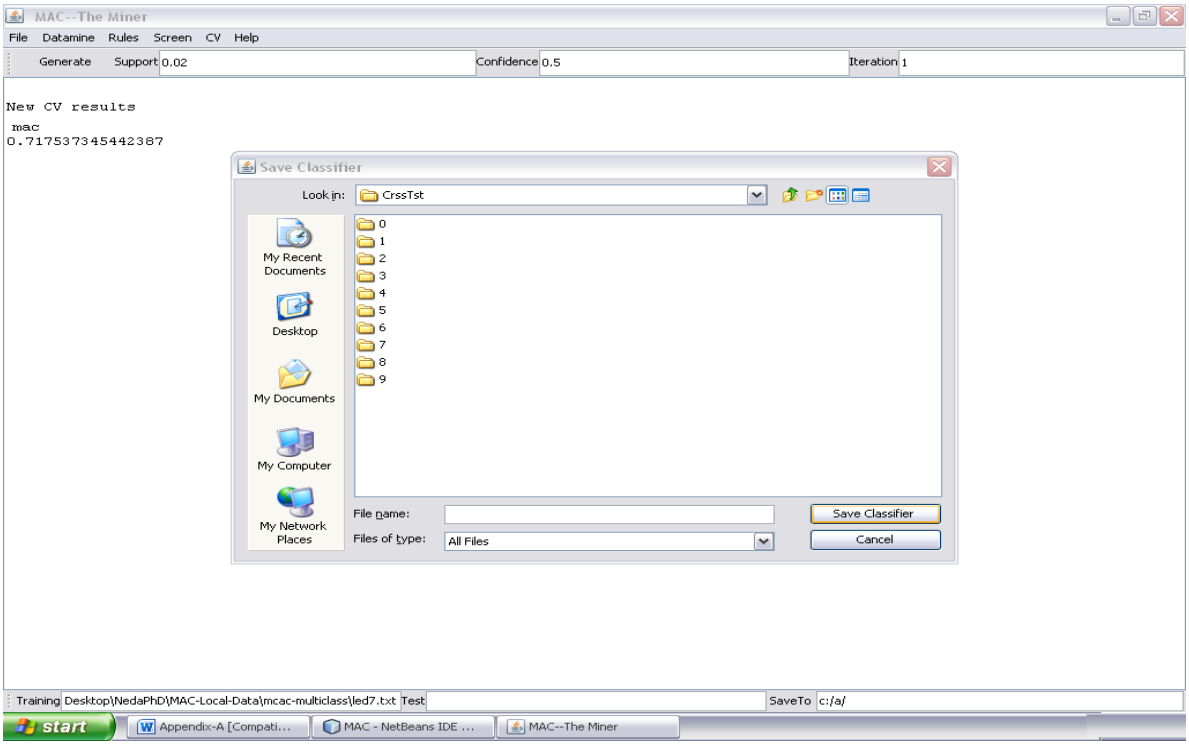

Fig. 11 Saving the outcome to an external file

Sample Results from Weka Software for other Rule based Classification Algorithms on the Same Data Set

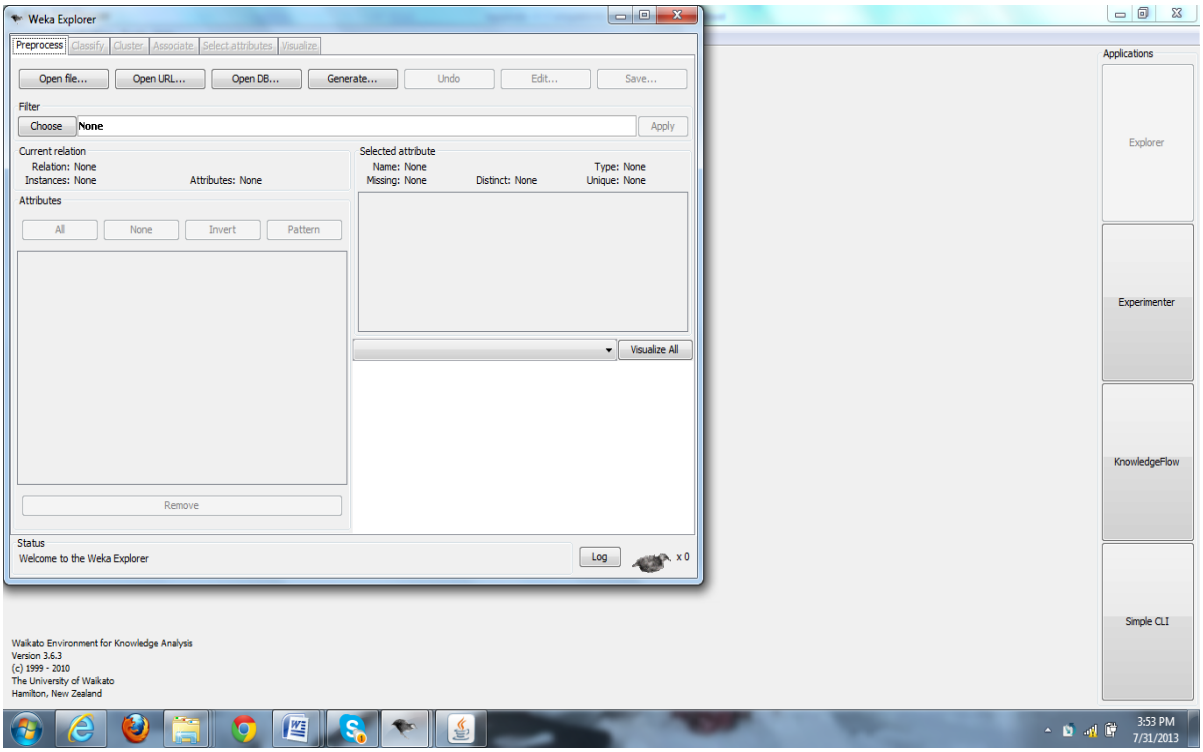

Fig. 12 The explorer GUI of Weka that has been used for experimentations

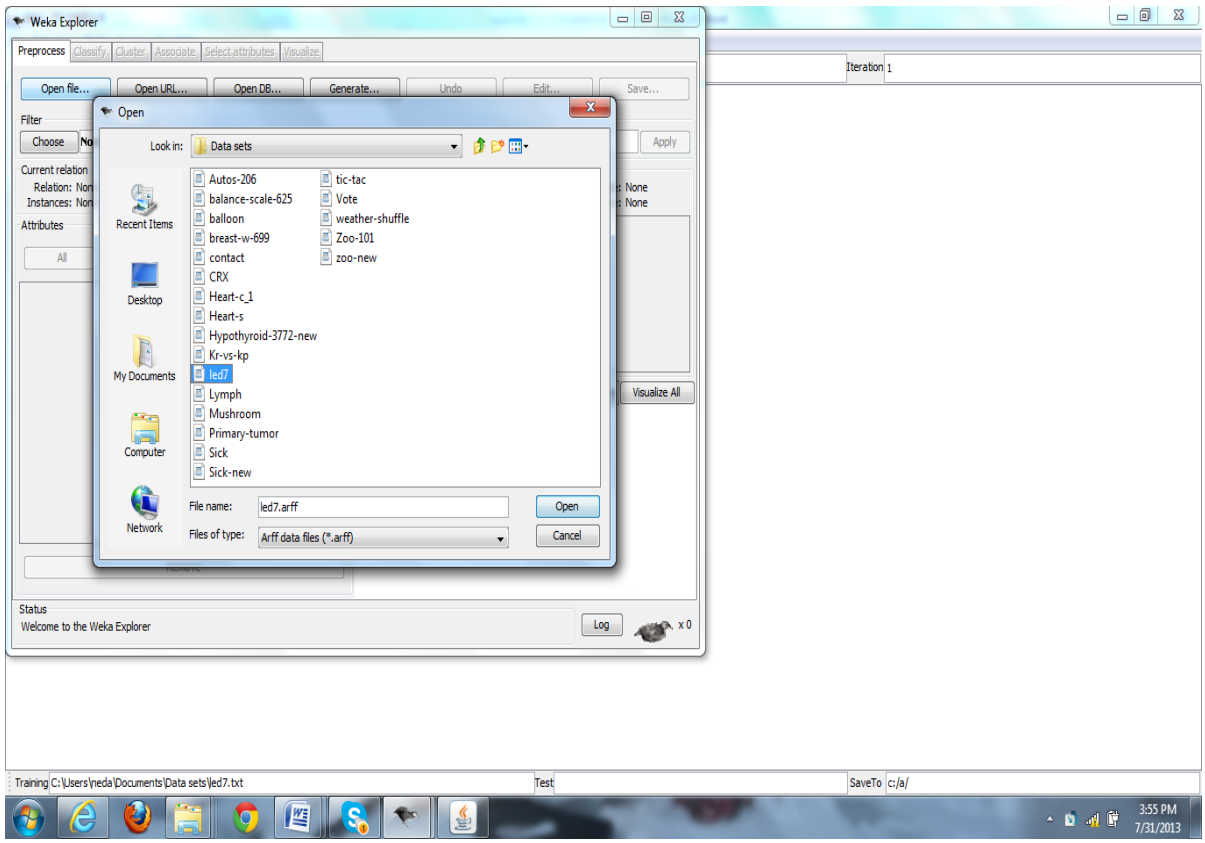

Fig. 13 Selecting the LED data set in Explorer

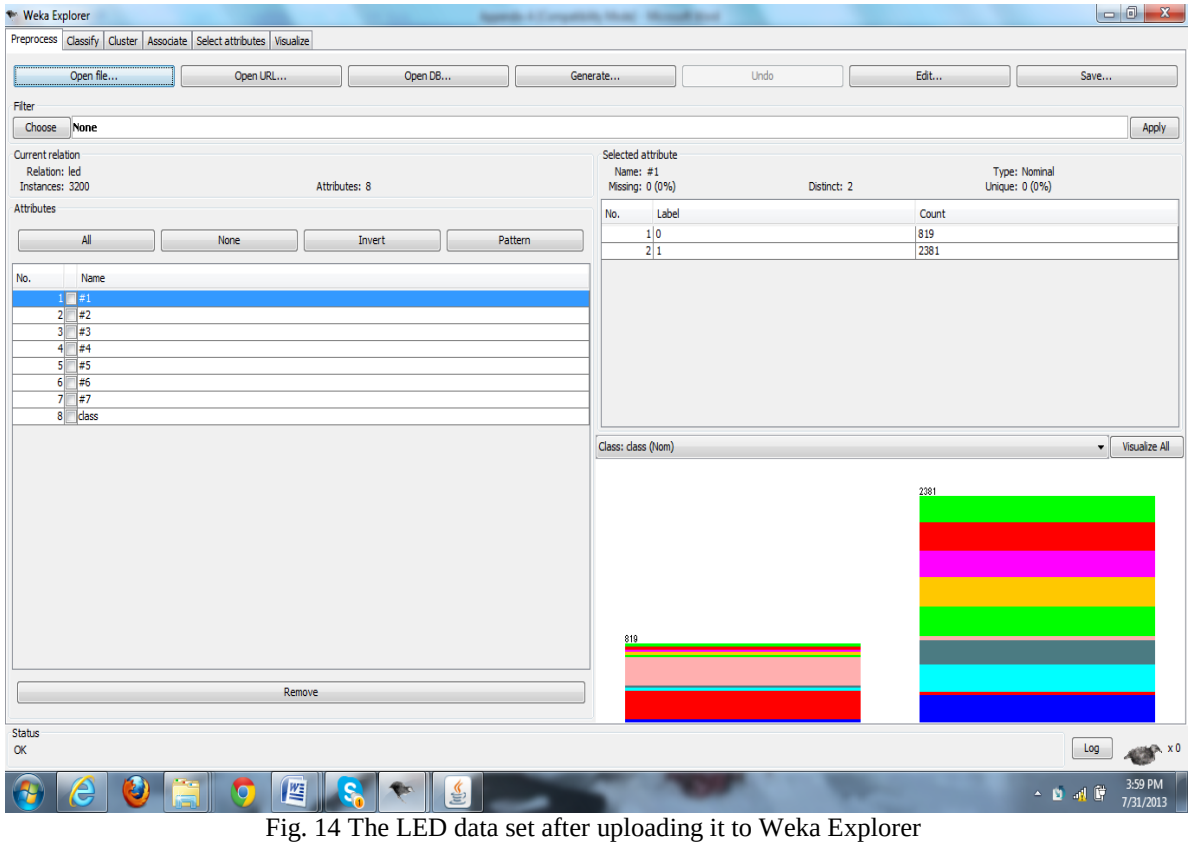

Fig. 14 The LED data set after uploading it to Weka Explorer

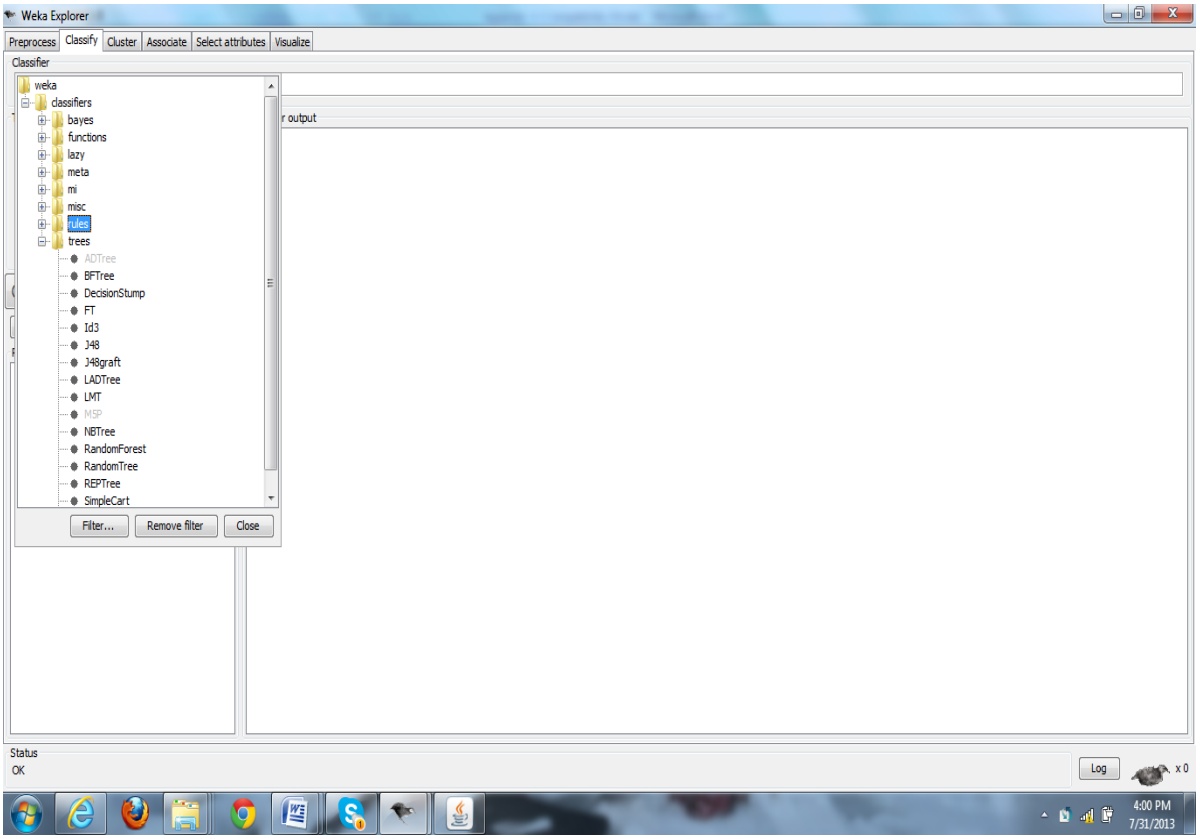

Fig. 15 Selecting C4.5 (J48 class name) algorithm from Trees structure in Weka

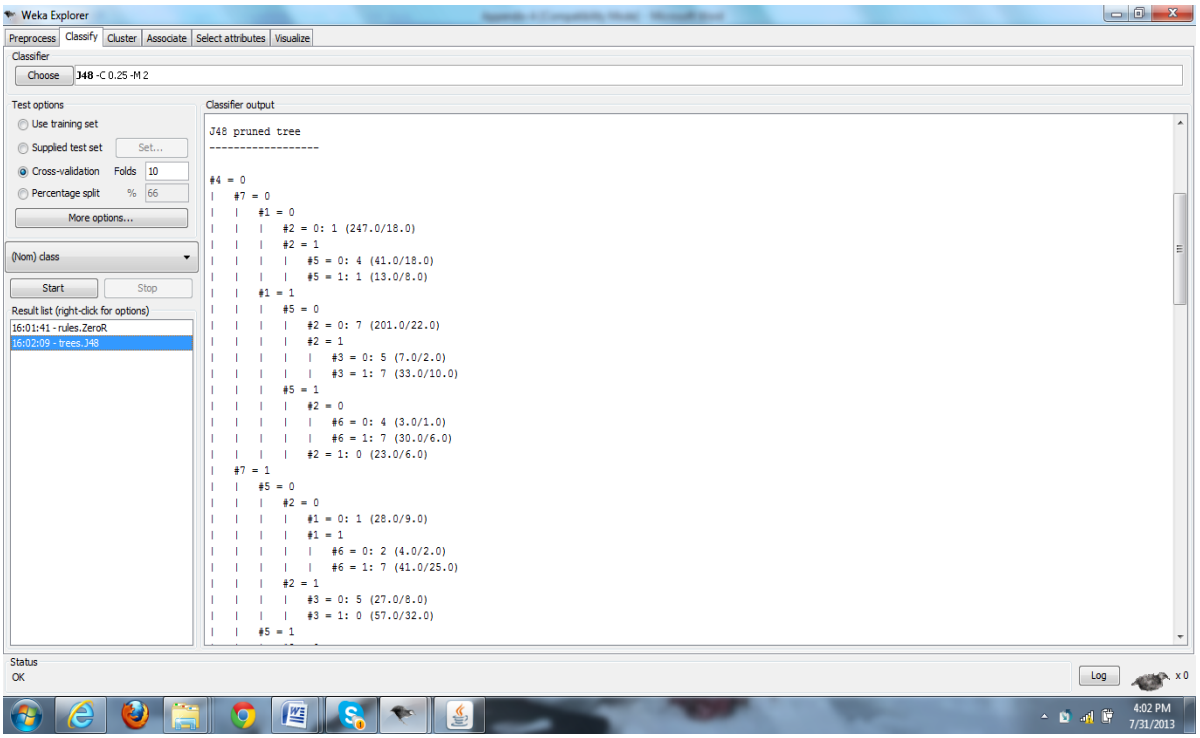

Fig. 16 Running C4.5 (J48) algorithm from Trees structure in Weka

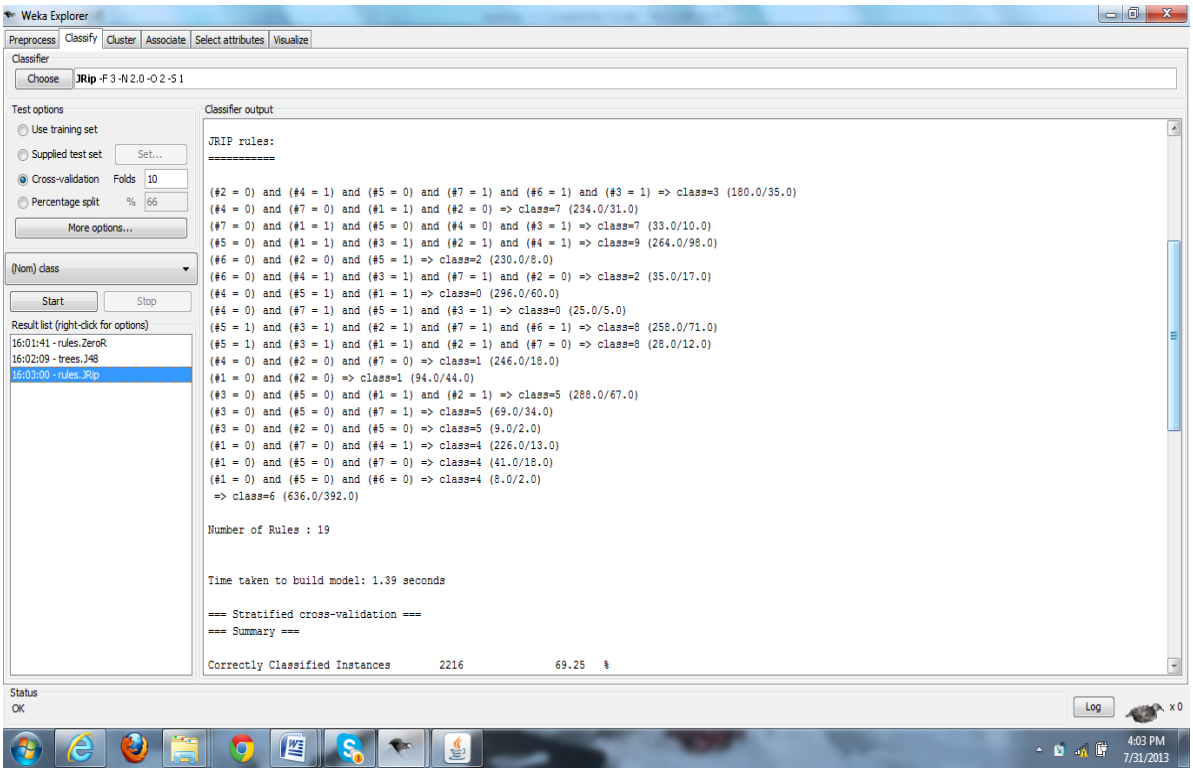

Fig. 17 Printing out the results of RIPPER algorithm (JRip class name) from Rules structure in Weka on the LED data set

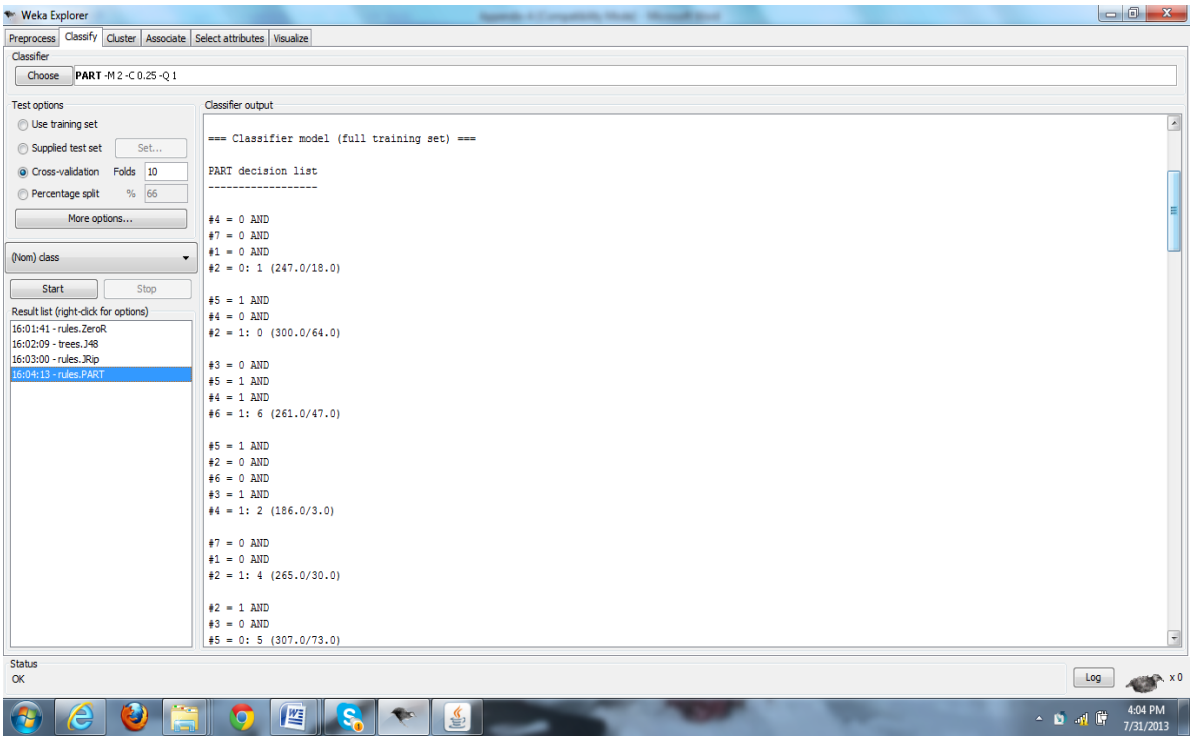

Fig. 18 Printing out the results of PART algorithm from Rules structure in Weka on the LED data set

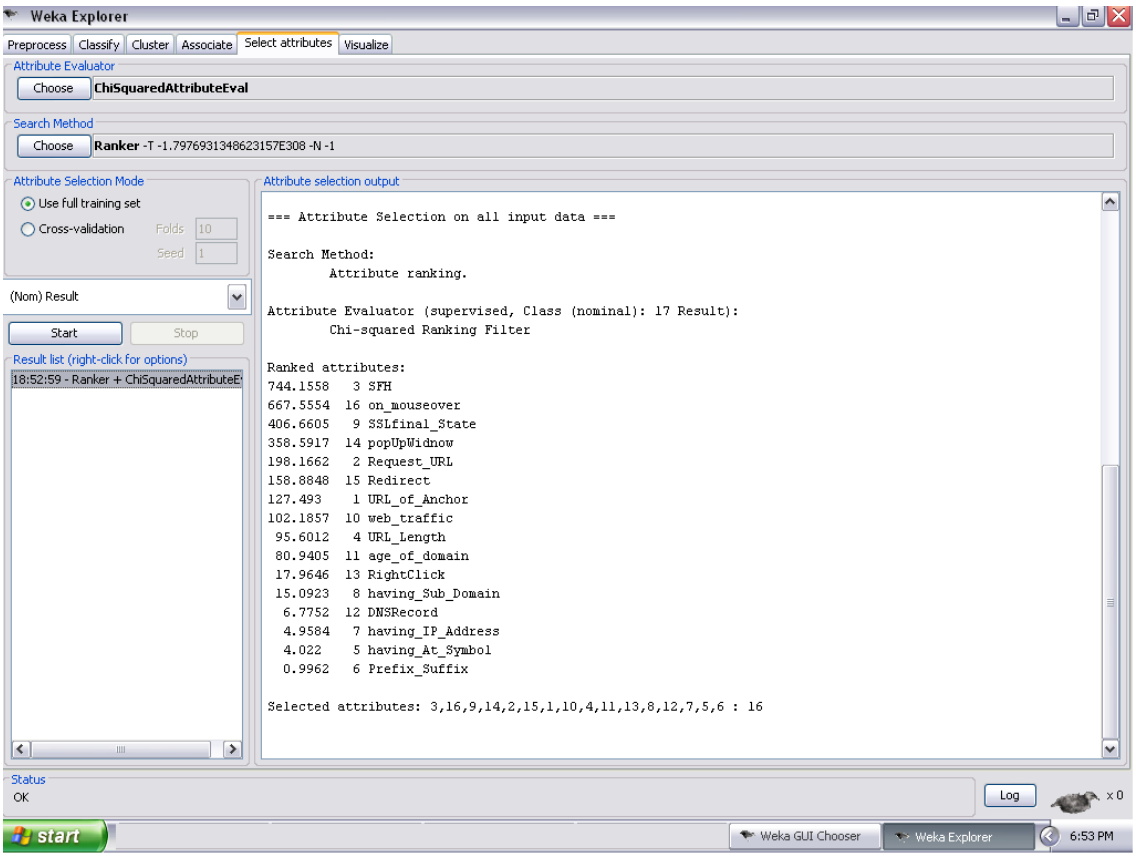

Fig. 19 Feature assessment on the phishing features set using Chi-Square Testing

Appendix B

Sample of Source Code

Title: DM

Description: AC classification project / Neda Abdelhamid

Copyright: Copyright (c) 2012

Company: DMU

Class MAC

```
package dm;
import javax.swing.UIManager;
import java.awt.*;

public class MAC {
 boolean packFrame = false;

 //Construct the application
 public MAC() {
  MainFrame frame = new MainFrame();
  //Validate frames that have preset sizes
  //Pack frames that have useful preferred size info, e.g. from their layout
  if (packFrame) {
   frame.pack();
  }
  else {
   frame.validate();
  }
  //Center the window
  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize = frame.getSize();
  if (frameSize.height > screenSize.height) {
   frameSize.height = screenSize.height;
  }
  if (frameSize.width > screenSize.width) {
   frameSize.width = screenSize.width;
  }
  frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
  frame.setVisible(true);
 }
```

```
 //Main method
 public static void main(String[] args) {
  try {
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
  }
  catch(Exception e) {
   e.printStackTrace();
  }
  new MAC();
 }
}
```

Class Cv // Cross Valdiation

```
package dm;
import java.util.*;
import java.io.*;
import javax.swing.*;

public class Cv {
 String fileName="";
 DataMine dm;
 DataMine dmTrn;
 DataMine dmTst;
 double support=0.0;
 double confidence=0.0;
 int iteration=0;
 final Set orgLines;
 public Cv(DataMine PDm){
  dm=PDm;
  orgLines=dm.entity.keySet();
 }
 public Cv(String PFileName) {
  fileName=PFileName;
  dm=new DataMine(PFileName);
  orgLines=dm.entity.keySet();
 }
 void setSuppConIter(double PSupport,double PConfidence,int PIteration){
  support=PSupport;
  confidence=PConfidence;
  iteration=PIteration;
 }
 Set getPortionTrn(double trnRate){
  Set resultSet=new HashSet();
  Iterator  iter=dm.classCol.items.entrySet().iterator();
  while(iter.hasNext()){
   Map.Entry e=(Map.Entry)iter.next();
   Set ts=(Set)e.getValue();
   int minLines=(int)Math.round(trnRate*ts.size()-0.5);
   if(minLines==0)minLines=1;
   Integer[] lns=new Integer[ts.size()];
   int dex=0;
   Iterator iter2=ts.iterator();
   while(iter2.hasNext()){
```

```
      lns[dex]=(Integer)iter2.next();
      dex++;
     }
     int[] linesIndexes=Tools.getRandomSamp(minLines,ts.size());
     for(int i=0; i<linesIndexes.length; i++){
      resultSet.add(lns[linesIndexes[i]]);
     }
    }

    return resultSet;
   }

   Set getRemainTst(Set trnSet){
    dm.allLines.removeAll(trnSet);
    return new HashSet(dm.allLines);
   }

   void setLines( Set PLines){
    dm.allLines=PLines;
   }

   void setDataMineLines(Set lines){

   }

   double[] getCvAccuracy(double PTrnRate,int repeat) throws IOException{
    double[] totalAccuracy=new double[5];
    for(int j=0; j<repeat; j++){
     dm.resetLines();
     dmTrn=new DataMine(fileName);
     dmTst=new DataMine(fileName);
     Set trainLines=getPortionTrn(PTrnRate);
     Set testLines=getRemainTst(trainLines);
//     System.out.print("\ntrainLine"+trainLines);
//     System.out.print("\ntestLine"+testLines);
     dmTrn.delLines(testLines);
     dmTst.delLines(trainLines);
//     JOptionPane.showMessageDialog(null,"\nbefore iterate , supp, conf
,iter,"+support+"\t"+confidence+"\t"+iteration+"\t");
     dmTrn.rules.iterate2(0.0,confidence,support*PTrnRate,iteration);
     dmTrn.rules.applyToDatamineAndSaveToOld(dmTst,"");
     double[] acc=dmTrn.rules.getAccuracy(dmTst);
     System.out.print("\n inside iteration Accuracy :"+ j+" :\t");
     for(int i=0; i<5; i++){
      System.out.print(""+acc[i]+"\t");
      totalAccuracy[i]+=acc[i];
     }
    }
    for(int k=0; k<5; k++){
      totalAccuracy[k]=(double)totalAccuracy[k]/(double)repeat;
    }
    return totalAccuracy;
   }
   public StringBuffer getLastClassifier()throws IOException{
    return dmTrn.rules.getClassifier();
```

```
 }
//////////new code for cv


 Set getPortionTrnNew(double trnRate,Set orgS){
  double correctRate=(double)orgLines.size()/(double)orgS.size();
  Set resultSet=new HashSet();
  Iterator  iter=dm.classCol.items.entrySet().iterator();
  while(iter.hasNext()){
   Map.Entry e=(Map.Entry)iter.next();
   Set ts=(Set)e.getValue();
//     orgS.removeAll(ts);//
   int minLines=(int)Math.round(trnRate*ts.size());//- 0.5*Math.random());
   ts.retainAll(orgS);
   if(minLines < 0)minLines=0;
   Integer[] lns=new Integer[ts.size()];
   int dex=0;
   Iterator iter2=ts.iterator();
   while(iter2.hasNext()){
    lns[dex]=(Integer)iter2.next();
    dex++;
   }
   if(minLines<ts.size()){
    int[] linesIndexes=Tools.getRandomSamp(minLines,ts.size());
    for(int i=0; i<linesIndexes.length; i++){
     resultSet.add(lns[linesIndexes[i]]);
    }
   }else{
    resultSet.addAll(ts);
   }
  }
  return resultSet;
 }

////////

 double[] getCvAccuracyNew(int numOfFolds,int repeat) throws IOException{
    Set remainLines;
    dm.resetLines();
    dmTrn=new DataMine(fileName);
    dmTst=new DataMine(fileName);
  ArrayList folds=new ArrayList(numOfFolds+10);
  final double PTstRate=(double)1.0/(double)numOfFolds;
  System.out.println("\n RATE inside getCvAccuracyNew IS "+PTstRate);

  double[] totalAccuracy=new double[5];
  for(int j=0; j<repeat; j++){
   remainLines=new HashSet(orgLines);
   folds.clear();
   System.out.println("\n remainLines iterate "+(j+1)+"= "+remainLines);
   for(int i=0; i<numOfFolds-1; i++){
    Set ts2=getPortionTrnNew(PTstRate,remainLines);
    remainLines.removeAll(ts2);
    folds.add(ts2);
    System.out.println("\n Test part "+(i+1)+"= "+ts2);
```

```
      System.out.println("\n remain part "+(i+1)+"= "+remainLines);
     }
     folds.add(remainLines);

     for(int k=0; k<numOfFolds; k++){
      dm.resetLines();
      Set testLines=(Set)folds.get(k);
      Set trainLines=new HashSet(orgLines);
      trainLines.removeAll(testLines);
//       System.out.print("\ntrainLine"+trainLines);
//       System.out.print("\ntestLine"+testLines);
//       dmTrn.setLines(testLines);////to be returned
//     dmTst.setLines(trainLines);////to be returned
/////sss
     dmTrn=new DataMine(fileName);
     dmTst=new DataMine(fileName);
     dmTrn.delLines(testLines);
     dmTst.delLines(trainLines);
/////sss
  //     JOptionPane.showMessageDialog(null,"\nbefore iterate , supp, conf
,iter,"+support+"\t"+confidence+"\t"+iteration+"\t");
       dmTrn.rules.iterate2(0.0,confidence,support*(1.0-PTstRate),iteration);
       dmTrn.rules.applyToDatamineAndSaveToOld(dmTst,"");
       double[] acc=dmTrn.rules.getAccuracy(dmTst);
       System.out.print("\n inside iteration Accuracy :"+ j+" :\t");
       for(int i=0; i<5; i++){
        System.out.print(""+acc[i]+"\t");
        totalAccuracy[i]+=acc[i];
       }

     }
    }
    for(int k=0; k<5; k++){
       totalAccuracy[k]=(double)totalAccuracy[k]/((double)repeat*(double)numOfFolds);
    }
    return totalAccuracy;
  }

}
```

Class Rules

```
package dm;
import java.util.*;
import java.io.*;
import javax.swing.*;
import java.text.*;


public class Rules {
  final int orgSize;
  int minOcc=0;
  int accOcc=0;
  int numOfCols;//number of fields without the class field
  private  DecimalFormat tt=new DecimalFormat("0000000000000000.0000");
```

```
 /**
  * Map mp (classifier)
  * key :String ruleId e.g. 20|*|47|
  * value: Rule type object
  */
 Map mp;
 /**
  * Arraylist al: contain the id of the survived rules of the ranked rule set.
  * the actual rules (Rule objects) exist in Map mp
  * items: ruleId added in a ranked order
  */
 ArrayList al;
 DataMine dm;
 public DataMine rulesDataMine;
 /**
  * rankedRuleSet TreeMap
  * key: hassh Code
  * value: long[]
  * long[0]: columnId (inside the datamine)
  * long[1]: itemId (inside the column)
  */
 public TreeMap rankedRuleSet;
/**
 * TreeMap pc (predicted class)
 * Key: Integer line ids of tested datamine
 * value:String the corrosponding rule ids in the classifier
 */
TreeMap pc=new TreeMap();///added when writing saveWithPredition() method
Set idis=new HashSet();//contains the original idis of the rules (new)
DataMine destDm;

 public Rules(DataMine dm2) {
   dm=dm2;
   orgSize=dm2.TOTAL_ENTITIES;
   numOfCols=dm.CLASS-1;
   mp=new HashMap();
   al=new ArrayList();
   rankedRuleSet=new TreeMap();
 }

 ///

 public String printRankedRules(){
  String s="\nthe rules";
  String s2="";
  Iterator itr =rankedRuleSet.entrySet().iterator();
  while(itr.hasNext()){
    Map.Entry e = (Map.Entry) itr.next();
    double dbl=((Double)e.getKey()).doubleValue();
    long[] a=(long[])e.getValue();
    int tint=(int)a[1];
    Column clmn=(Column)dm.existingColumns.get(new Long(a[0]));
    s2+="\n"+tt.format(dbl)+"\t"+a[0]+"\t"+a[1]+"\t"+clmn.calculateItemConfidence(new
Integer(tint))+"\n";//+clmn.prntRuleOcc(new Integer(tint));//tt.format
  }
```

```
  return s+s2;
 }


 ////

 public void rankARule(int occ,int nomin, long columnId, int rowId,String pClass){
 System.out.println("rank \tcolId= "+columnId+"\trowId="+rowId+"\tpClass "+pClass);
  rankARule( occ, nomin,  columnId, rowId);
 }

 public void rankARule(int occ,int nomin, long columnId, int rowId){
  double[] b={(double)nomin/occ,
          (double)nomin/orgSize,
          (double)1.0-(double)columnName.length(columnId)/dm.allColumns,
          (double)occ/orgSize,
          (double)((double)dm.allColumns-(double)columnId)/(double)dm.allColumns,
          (double)rowId/orgSize};
 double a[]={3,2,3,3,2};
 double hCode=Tools.ruleOrder(a,b);
 long[] rl=new long[2];
 rl[0]=columnId;
 rl[1]=rowId;

//  System.out.println("rank rule Code="+tt.format(hCode)+"\tcolId= "+columnId+"\trowId="+rowId);
 rankedRuleSet.put(new Double(-hCode),rl);

 }

 public boolean addRule(long clmn2,int itm2,int occ2,String cls){
  String[] cols=new String[numOfCols];
  String[] clss=new String[100];
  clss[0]=cls;//this is not good ,but to be changed later
  int[] clssOcc=new int[100];
  clssOcc[0]=occ2;
  int allOcc=occ2;
  String ruleId=fillCols(cols,clmn2,itm2);
  addRule(new Rule(numOfCols,
              cols,
              allOcc,
              1,
              clss,
              clssOcc));
  return true;
 }
  private String fillCols(String[] cols,long clm, int itm){
  String s="";
  for (byte i=0; i<numOfCols; i++){
   if (((1L<<i) & clm) != 0){
    cols[i]=((String[])dm.entity.get(new Integer(itm)))[i+1];
   }else{
    cols[i]="*";
   }
   s+=cols[i]+"|";
  }
```

```
  return s;
 }
 public boolean addRule(Rule rl){
  String s=rl.getId();
  Rule trl=(Rule)mp.get(s);
  if(trl==null){
    mp.put(s,rl);
    al.add(s);
  }else{
   trl.updateRule(rl);
   mp.put(s,trl);
  }
  return true;
 }

 public void SaveToFile(String fs)throws IOException{
  int tot=mp.keySet().size();
  BufferedWriter out2= new BufferedWriter(new FileWriter(fs));
  out2.write(getClassifier().toString());
  out2.close();

 }
 public StringBuffer getClassifier(){
  int tot=mp.keySet().size();
  StringBuffer out2= new StringBuffer();
  out2.append("number of rules\t"+tot+"\n");
  out2.append("number of columns\t"+numOfCols+"\n");
  out2.append("original dataset size\t"+orgSize+"\n");
  out2.append("minimum occurance\t"+minOcc+"\n");
  for(int i=0; i<numOfCols;i++){
    out2.append("Co1_"+i+"\t");
  }
  out2.append("allOcc\t#clss\n");
  for(int j=0; j<al.size(); j++){
    Rule rl=(Rule)mp.get((String)al.get(j));
    for(int i=0; i<numOfCols; i++){
      out2.append(rl.cols[i]+"\t");
    }
    out2.append(rl.allOcc+"\t");
    out2.append(rl.numOfClasses+"\t");
    for(int i=0; i<rl.numOfClasses; i++){
      out2.append(rl.clss[i]+"\t");
    }
    for(int i=0; i<rl.numOfClasses; i++){
      out2.append(rl.clssOcc[i]+"\t");
    }
    out2.append("\n");
  }
  out2.append("Accuracy  is\n"+
            accOcc+"/"+orgSize+"= "+(double)accOcc/orgSize);
  return out2;
 }

public void loadFromFile(String fs)throws IOException{
}
```

```
public void applyToDatamineAndSaveTo(DataMine testDataMine,String resultFile)throws IOException{
}
public void saveWithPrediction(String resultFile)throws IOException{
  double countSingleClss=0;
  double countMultiSched=0;
  double countMultimac=0;
  double countMultimac1=0;
  double countSingleBest=0;

  BufferedWriter out2= new BufferedWriter(new FileWriter(resultFile));
  out2.write("add your comments here\n");
  out2.write("line\t");
  for(int i=1; i<=numOfCols; i++){
    out2.write("C"+i+"\t");
  }
  out2.write("class\t");
  out2.write("bClass\t");
  out2.write("single\t");
  out2.write("sched\t");
  out2.write("mac\t");
  out2.write("mac1\t");
  out2.write("bClass\t");
  out2.write("-->\t");
  out2.write("Multi Predicted Classes\n");


  Iterator iter=destDm.entity.entrySet().iterator();
  while(iter.hasNext()){
    Map.Entry e=(Map.Entry)iter.next();
    Integer ti=(Integer)e.getKey();
    out2.write(ti.intValue()+"\t");
    String[] a2=(String[])e.getValue();
   for(int i=1; i<=numOfCols; i++){
     out2.write(a2[i]+"\t");
    }
   out2.write(a2[0]+"\t");//write the org. class
   Rule rl=(Rule)pc.get(ti);
   //Rule rl=(Rule)mp.get(rlId);
   String bestClass=rl.clss[0];
   int bestOcc=rl.clssOcc[0];
   int bestJi=0;
   for(int ji=1; ji<rl.numOfClasses;ji++){
     if(rl.clssOcc[ji]>bestOcc){
       bestClass=rl.clss[ji];
       bestOcc=rl.clssOcc[ji];
       bestJi=ji;
     }
   }
   boolean b=false;
   double dd=0;
   double singleClss=0;
   double muliSched=0;
   double multimac=0;
   double multimac1=0;
   double singleBest=0;
```

```
  for(int j=0; j<rl.numOfClasses; j++){
    if(a2[0].equals(rl.clss[j])){
      if(j==0){
        singleClss=1.0;
        countSingleClss+=singleClss;
      }
      muliSched=1.0;
      countMultiSched+=muliSched;
      multimac=(double)rl.clssOcc[j]/rl.clssOcc[0];
      countMultimac+=multimac;
      multimac1=(double)rl.clssOcc[j]/rl.allOcc;
      countMultimac1+=multimac1;
      if(j==bestJi){
        singleBest=1.0;
        countSingleBest+=singleBest;
      }
      break;
    };
  }
  out2.write(""+bestClass+"\t");
  out2.write(""+singleClss+"\t"+muliSched+"\t"+multimac+"\t"+multimac1+"\t"+singleBest+"\t");
  out2.write("\t");
  int space=12;
  for(int j=0; j<rl.numOfClasses; j++){
    out2.write(rl.clss[j]+"\t");
    space--;
  }

  for(;space>0;space--)
    out2.write("\t");
  for(int j=0; j<rl.numOfClasses; j++){
     out2.write(rl.clssOcc[j]+"\t");
  }
  out2.write("\n");
  }
  out2.write("singlePred\t");  out2.write(""+countSingleClss+"\n");
  out2.write("multiSched\t");  out2.write(""+countMultiSched+"\n");
  out2.write("multimac\t");   out2.write(""+countMultimac+"\n");
  out2.write("multimac1\t");  out2.write(""+countMultimac1+"\n");
  out2.write("singleBest\t");  out2.write(""+countSingleBest+"\n");
  out2.close();
}
public void applyToDataMineFileAndSaveTo(String testFileName,String resultFile)throws IOException{
   DataMine testDataMine =new DataMine(testFileName);
   applyToDatamineAndSaveTo(testDataMine,resultFile);
}
public void applyToDataMineFileAndSaveTo2(String testFileName,String resultFile)throws IOException{
   DataMine testDataMine =new DataMine(testFileName);
   applyToDatamineAndSaveTo2(testDataMine,resultFile);

}
 /**
  * build the classifier
  * @return
  * @throws IOException
```

```
  */
  public Set checkRules() throws IOException{
   if(rankedRuleSet.size()==0)return new HashSet();
   Set deletedLines=new HashSet();
   int sz=0;
   Iterator itr=rankedRuleSet.entrySet().iterator();
   Map.Entry e=(Map.Entry)itr.next();
   long[] a=(long[])e.getValue();
//    while (itr.hasNext()) {
//      Map.Entry e=(Map.Entry)itr.next();
//      long[] a=(long[])e.getValue();
     Column clmn=(Column)dm.existingColumns.get(new Long(a[0]));
     Sccl itm=(Sccl)clmn.items.get(new Integer((int)a[1]));
     String cls=itm.classId;
     TreeSet ts=new TreeSet(itm.lines);
     ts.retainAll((Set)dm.classCol.items.get(cls));
     ts.removeAll(deletedLines);
     if(ts.size()==0)return new HashSet();
     addRule(a[0],(int)a[1],ts.size(),cls);
     deletedLines.addAll(ts);
//    }

   accOcc+=deletedLines.size();//+addDefaultClass(deletedLines)

   return deletedLines;
  }

  public Set checkRules2() throws IOException{
   Set deletedLines=new HashSet();
   int sz=0;
   //to test the order of the ranked rules
  System.out.println(printRankedRuls());
//    System.out.println("inside Ckeck rules ,All lines :"+ dm.allLines);
   // end test
   Iterator itr=rankedRuleSet.entrySet().iterator();
   while (itr.hasNext()) {
     Map.Entry e=(Map.Entry)itr.next();
     long[] a=(long[])e.getValue();
     //JOptionPane.showMessageDialog(null,"ranked col" +a[0]+" "+a[1]);
     Column clmn=(Column)dm.existingColumns.get(new Long(a[0]));
     Sccl itm=(Sccl)clmn.items.get(new Integer((int)a[1]));
     //JOptionPane.showMessageDialog(null,"clmnId" +clmn.columnId);
     //JOptionPane.showMessageDialog(null,"itm lines " +itm.lines);
     String cls=itm.classId;
     TreeSet ts=new TreeSet(itm.lines);
     ts.retainAll((Set)dm.classCol.items.get(cls));
     ts.removeAll(deletedLines);
     if(ts.size()==0)continue;
     addRule(a[0],(int)a[1],ts.size(),cls);
     //JOptionPane.showMessageDialog(null,"a"+a[0]+"\t"+a[1]);
     deletedLines.addAll(ts);
   }
    //JOptionPane.showMessageDialog(null,"DELETED lINES ="+ deletedLines.size());

   accOcc+=deletedLines.size();//+addDefaultClass(deletedLines)
```

```
  return deletedLines;
 }


 public int addDefaultClass(Set ts2){

  Map tm= new HashMap();
  Set ts=new HashSet(dm.allLines);
  ts.removeAll(ts2);
  Iterator itr=ts.iterator();
  while( itr.hasNext()){
   String stcls=dm.classArray[((Integer)itr.next()).intValue()];
   Integer freq=(Integer)tm.get(stcls);
   tm.put(stcls,( freq==null? new Integer(1):new Integer(freq.intValue()+1)));
  }
  // calculate the max confidence
  String maxClass=" ";
  int maxInt=0;
  for (Iterator i=tm.entrySet().iterator(); i.hasNext(); ) {
   Map.Entry e = (Map.Entry) i.next();
   int j=((Integer)e.getValue()).intValue() ;
   if( j > maxInt){
    maxInt=j;
    maxClass=(String)e.getKey();
   }
  }
  String[] cols=new String[numOfCols];
  for(int j=0; j<numOfCols; j++){
   cols[j]="*";
  }
  int allOcc=maxInt;
  int numOfClasses=1;
  String[] clss=new String[100];
  clss[0]=maxClass;
  int[] clssOcc=new int[100];
  clssOcc[0]=maxInt;
  Rule rl=new Rule(numOfCols,
                cols,
                allOcc,
                numOfClasses,
                clss,
                clssOcc);
  addRule(rl);
  return maxInt;
 }
 public void iterate2(double minRemainInst,double conf, double supp,int numOfItr)
  throws IOException{
//  JOptionPane.showMessageDialog(null,"iterate "+numOfItr);
  minOcc=(int)Math.round(orgSize*supp+0.5);
  int minRemainInstOcc=(int)Math.round(orgSize*minRemainInst+0.5);
  System.out.print("\nMInimum (inside Iterate)"+minOcc);
  int RemainInstOcc=orgSize;
  Set deletedRows=new HashSet();
  int iter=0;
```

```
   while(iter<numOfItr){
     iter++;
     int te=RemainInstOcc;
     rankedRuleSet.clear();
     dm.generateColumns(supp,conf);
     deletedRows=checkRules2();
     RemainInstOcc-=deletedRows.size();
     dm.delLines(deletedRows);
    }
    addDefaultClass(deletedRows);
  }

  public void iterate(double minRemainInst,double conf, double supp,int numOfItr)
  throws IOException{
//   JOptionPane.showMessageDialog(null,"iterate "+numOfItr);
   minOcc=(int)Math.round(orgSize*supp+0.5);
   int minRemainInstOcc=(int)Math.round(orgSize*minRemainInst);
   int RemainInstOcc=orgSize;
   Set deletedRows=new HashSet();
   int iter=1;
   int te=RemainInstOcc;
   ///
   idis.clear();
   rankedRuleSet.clear();
   dm.generateColumns(supp,conf);
   deletedRows=checkRules();
   RemainInstOcc-=deletedRows.size();
   dm.delLines(deletedRows);
   int rem=dm.TOTAL_ENTITIES;
   System.out.println(" 1  lines remained :"+ rem+ "\titeration :"+iter+"\trankedRules
:"+rankedRuleSet.size());
   if(minRemainInstOcc >=dm.TOTAL_ENTITIES){
    JOptionPane.showMessageDialog(null,"l   ast iteratiom "+iter);
   }
   deletedRows.clear();
   ///
    while(true){
     iter++;
     rankedRuleSet.clear();
     dm.generateColumnsNew();
     deletedRows=checkRules();
     RemainInstOcc-=deletedRows.size();
     dm.delLines(deletedRows);
     rem=dm.TOTAL_ENTITIES;
     System.out.println("lines remained :"+ rem+ "\titeration :"+iter+"\trankedRules :"+rankedRuleSet.size());
     if(minRemainInstOcc >=dm.TOTAL_ENTITIES){
       JOptionPane.showMessageDialog(null,"last iteratiom "+iter);
       break;
     }
     deletedRows.clear();
//     if (te==RemainInstOcc)return;
    }
 //   accOcc+=
     System.out.print("\nRemain lines"+dm.allLines);
    addDefaultClass(deletedRows);
```

```
 }

 void setRulesDataMine(){
  Map entts=new HashMap(mp.size()+100);
  String[] aClass=new String[mp.size()+1] ;
  for(int i=0; i< al.size(); i++){
   String[] row=new String[numOfCols+1];
   Rule rl=(Rule)mp.get((String)al.get(i));
   System.arraycopy(rl.cols,0,row,1,numOfCols);
   row[0]=rl.getId();
   aClass[i+1]=rl.getId();
   entts.put(new Integer(i+1),row);
  }
  rulesDataMine=new DataMine(entts,aClass);
  rulesDataMine.generateOccuranceColumns();
 }
 public void applyToDatamineAndSaveTo2(DataMine testDm,String desFile)throws IOException{
  setRulesDataMine();
  pc.clear();
//   System.out.println(rulesDataMine.printDataMine());
  int counter=0;
  Iterator iter=testDm.entity.entrySet().iterator();
  while(iter.hasNext()){
   Map.Entry e=(Map.Entry)iter.next();
   TreeSet remainRules=new TreeSet(rulesDataMine.entity.keySet());
   String[] row=(String[])e.getValue();
   String condit="";
   for(int cd=1;cd<=numOfCols; cd++){//for test to be deleted later
    condit+=row[cd]+" ";
   }
   for(int i=1; i<=numOfCols; i++){
    long intCol=(long)Math.pow(2,i-1);
    TreeSet ts=new TreeSet();
    ts.addAll((Set)rulesDataMine.getValueOccurancesInColumn(row[i],intCol));
    ts.addAll((Set)rulesDataMine.getValueOccurancesInColumn("*",intCol));
    remainRules.retainAll(ts);
   }
   Rule rl=pickARule(remainRules);
   pc.put((Integer)e.getKey(),rl);
   counter++;
  }
  destDm=testDm;
 }
/*  Rule pickARule(TreeSet PRemainRules){
  ///
  int lastDefaultLine=-1;
  if(PRemainRules.size()>1)
   lastDefaultLine=((Integer)PRemainRules.last()).intValue();
//   System.out.println("remain new:"+PRemainRules);
  Map tm= new HashMap();
  Iterator itr=PRemainRules.iterator();
  while( itr.hasNext()){
   int line=((Integer)itr.next()).intValue();
   if(line==lastDefaultLine)continue;
```

```
    String ruleId=(String)al.get(line-1);
    Rule rl=(Rule)mp.get(ruleId);
 //    String stcls;
    for(int i=0; i<rl.numOfClasses; i++){
      String stcls=rl.clss[i];
      Integer freq=(Integer)tm.get(stcls);
      if(freq==null){
        tm.put(stcls,new Integer(rl.clssOcc[i]));
      }else{
        tm.put(stcls,new Integer(freq.intValue()+rl.clssOcc[i]));
      }
    }
  }
  // calculate the max class
  String[] sortClass=new String[tm.size()] ;
  int[] sortOcc=new int[tm.size()];
  //fill array with the values from tm
  int allOcc=0;
  int index=0;
  Iterator iter=tm.entrySet().iterator();
  while(iter.hasNext()){
    Map.Entry e=(Map.Entry)iter.next();
    sortClass[index]=(String)e.getKey();
    sortOcc[index]=((Integer)e.getValue()).intValue();
    allOcc+=((Integer)e.getValue()).intValue();
    index++;
  }
  /// sort according to the occurrences
  for(int i=0; i<sortOcc.length; i++){
    for(int j=0; j<sortOcc.length-1;j++){
      if(sortOcc[j+1]>sortOcc[j]){
        String tempS=sortClass[j];
        sortClass[j]=sortClass[j+1];
        sortClass[j+1]=tempS;
        int tempI=sortOcc[j];
        sortOcc[j]=sortOcc[j+1];
        sortOcc[j+i]=tempI;
      }//end if
    }//end for 1
  }//enf for 2
  Rule resultRule=new Rule(numOfCols,new String[numOfCols],allOcc,
                           sortClass.length,sortClass,sortOcc);
  return resultRule;
 }
*/
 public StringBuffer returnPredictedClass(DataMine dm3)throws IOException{
 return new StringBuffer();
 }
 public StringBuffer applyToDatamine(DataMine testDataMine)throws IOException{
  return new StringBuffer();
 }
 public void applyToDatamineAndSaveToOld(DataMine testDm,String desFile)throws IOException{
  setRulesDataMine();
  pc.clear();
 //   System.out.println(rulesDataMine.printDataMine());
```

```
   int counter=0;
   Iterator iter=testDm.entity.entrySet().iterator();
   while(iter.hasNext()){
    Map.Entry e=(Map.Entry)iter.next();
    TreeSet remainRules=new TreeSet(rulesDataMine.entity.keySet());
    String[] row=(String[])e.getValue();
    String condit="";
    for(int cd=1;cd<=numOfCols; cd++){//for test to be deleted later
     condit+=row[cd]+" ";
    }
    for(int i=1; i<=numOfCols; i++){
     long intCol=(long)Math.pow(2,i-1);
     TreeSet ts=new TreeSet();
     ts.addAll((Set)rulesDataMine.getValueOccurancesInColumn(row[i],intCol));
     ts.addAll((Set)rulesDataMine.getValueOccurancesInColumn("*",intCol));
     remainRules.retainAll(ts);
    }
    Rule rl=pickARuleOld(remainRules);
    pc.put((Integer)e.getKey(),rl);
    counter++;
   }
   destDm=testDm;

  }
  public void applyToDataMineFileAndSaveToOld(String testFileName,String resultFile)throws IOException{
   DataMine testDataMine =new DataMine(testFileName);
   applyToDatamineAndSaveToOld(testDataMine,resultFile);

  }
  Rule pickARule(TreeSet PRemainRules){
   int line=((Integer)PRemainRules.first()).intValue();
   String ruleId=(String)al.get(line-1);
   Rule rl=(Rule)mp.get(ruleId);
   return rl;
  }
  Rule pickARuleOld(TreeSet PRemainRules){
   int line=((Integer)PRemainRules.first()).intValue();
   String ruleId=(String)al.get(line-1);
   Rule rl=(Rule)mp.get(ruleId);
   return rl;
  }
 ///
  double[] getAccuracy(DataMine testDataMine){
   double[] resultArray=new double[5];
   double countSingleClss=0;
   double countMultiSched=0;
   double countMultimac=0;
   double countMultimac1=0;
   double countSingleBest=0;
   Iterator iter=testDataMine.allLines.iterator();
   while(iter.hasNext()){
    Integer lineNum=(Integer)iter.next();
    String[] LineValues=(String[])testDataMine.entity.get(lineNum);
    Rule rl=(Rule)pc.get(lineNum);
   //Rule rl=(Rule)mp.get(rlId);
```

```
    String bestClass=rl.clss[0];
    int bestOcc=rl.clssOcc[0];
    int bestJi=0;
    for(int ji=1; ji<rl.numOfClasses;ji++){
     if(rl.clssOcc[ji]>bestOcc){
       bestClass=rl.clss[ji];
       bestOcc=rl.clssOcc[ji];
       bestJi=ji;
     }    }
    boolean b=false;
    double dd=0;
    double singleClss=0;
    double muliSched=0;
    double multimac=0;
    double multimac1=0;
    double singleBest=0;
    String currentClass=LineValues[0];
    for(int j=0; j<rl.numOfClasses; j++){
     if(currentClass.equals(rl.clss[j])){
       if(j==0){
         singleClss=1.0;
         countSingleClss+=singleClss;
       }
       muliSched=1.0;
      countMultiSched+=muliSched;
       multimac=(double)rl.clssOcc[j]/rl.clssOcc[0];
       countMultimac+=multimac;
       multimac1=(double)rl.clssOcc[j]/rl.allOcc;
       countMultimac1+=multimac1;
       if(j==bestJi){
         singleBest=1.0;
         countSingleBest+=singleBest;
       }
       break;
     }    }   }
   double allLinesSize=(double)testDataMine.allLines.size();
   resultArray[0]=countSingleClss/allLinesSize;
   resultArray[1]=countMultiSched/allLinesSize;
   resultArray[2]=countMultimac/allLinesSize;
   resultArray[3]=countMultimac1/allLinesSize;
   resultArray[4]=countSingleBest/allLinesSize;
   return resultArray;
  }

}
```